Fachhochschule Köln
Cologne University of Applied Sciences

Campus Gummersbach
Fakultät für Informatik und Ingenieurwissenschaften

Fachhochschule Dortmund
University of Applied Sciences and Arts

Fachbereich Informatik

Verbundstudiengang Wirtschaftsinformatik

Diplomarbeit zur Erlangung des Diplomgrades
Diplom-Informatiker (FH)
in der Fachrichtung Informatik

„Konzeption und Implementierung eines PL/SQL Trainers auf Basis von JSP und eLML“

Erstprüfer:	Prof. Dr. Heide Faeskorn-Woyke
Zweitprüfer:	Prof. Dr. Birgit Bertelsmeier
vorgelegt am:	30.05.2011
von cand.	Markus Rechs
aus	Theodor-Lövenich-Str. 12
	50226 Frechen
Tel.:	02234/4357706
eMail:	markus.rechs@gmx.de
Matr.-Nr.:	11047398

Markenrechtlicher Hinweis

Die in dieser Arbeit wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenzeichen usw. können auch ohne besondere Kennzeichnung geschützte Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Kurzfassung

Die Diplomarbeit „Konzeption und Implementierung eines PL/SQL Trainers auf Basis von JSP und eLML“ beschäftigt sich mit der Erstellung von eLearning Anwendungen, für das Themengebiet PL/SQL, mit speziellem Fokus auf die Programmierung von Datenbanktriggern. Im theoretischen Teil der Arbeit wird das Thema eLML ausführlich behandelt und dabei die Funktionsweise sowie die Hintergrundgeschichte des eLML-Frameworks erläutert. Der praktische Teil der Arbeit unterteilt sich in zwei Abschnitte, im Ersten Abschnitt wird beschrieben wie bestehende Lerneinheiten zum Thema PL/SQL aus dem MS-Power-Point Format in das eLML Format übertragen werden, wobei die praktische Anwendung von eLML dargestellt wird. Im zweiten Abschnitt wird die Konzeption und die Implementierung einer JSP-Webanwendung beschrieben, die eine interaktive Trainingsanwendung für die Programmierung von Triggern mit PL/SQL darstellt.

Abstract

The diploma thesis „Konzeption und Implementierung eines PL/SQL Trainers auf Basis von JSP und eLML“ concerns itself with the production of eLearning applications for the topic area PL/SQL with special focus on the programming of data base triggers. In the theoretical part of the thesis the topic eLML is treated in detail and the functionality as well as the background history of the eLML Frameworks is described. The practical part of the thesis is divided into two sections, the first section describes how existing learning units about PL/SQL are transferred from the MS PowerPoint format into the eLML format, whereby the practical use of eLML is represented. In the second section the conception and the implementation of a JSP Webanwendung is described, which represents an interactive training- application for the programming of triggers with PL/SQL.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CSS .....Cascading Style Sheets

DBA.....Database Administrator

DBS.....Datenbanksystem

DML.....Data Manipulation Language

DuI.....Datenbanken und Informationssysteme

DTD.....Document Type Definition

edb .....eLearning Datenbank Portal

eLML .....eLesson Markup Language

GITTA.....Geographic Information Technology Training Alliance

HTTP.....Hypertext Transfer Protocol

JDBC .....Java Database Connectivity

JNDI.....Java Naming and Directory Interface

JSP.....Java Server Pages

MVC.....Model-View-Controller

ODF.....Open Document Format

PL/SQL.....Procedural Language/Structured Query Language

SQL .....Structured Query Language

SVC.....Swiss Virtual Campus

WYSIWYG.....What You See Is What You Get

XHTML .....Extensible HyperText Markup Language

XML.....Extensible Markup Language

XSL .....Extensible Stylesheet Language

XSLT.....Extensible Stylesheet Language Transformation

YAML.....Yet Another Multicolumn Layout

1 Einleitung

Die FH-Köln betreibt am Institut für Informatik, für das Studienfach DuI, das eLearning Portal edb. Dieses Portal enthält bereits diverse eLearning Angebote für die, in diesem Kurs, behandelten Themengebiete. Da im Rahmen des Fachs DuI auch das Thema PL/SQL intensiv behandelt wird und das edb hierzu bisher keinerlei eLearning Angebote beinhaltet, sollte das edb, im Rahmen dieser Diplomarbeit, um entsprechende eLearning Inhalte zu diesem Themengebiet erweitert werden.

Eine der Erweiterungen sollte es sein, die zum Thema PL/SQL bereits bestehenden Foliensätze, vorliegende im MS-Power-Point Format, in das HTML Format zu übertragen und somit deren Inhalte in das edb einbinden zu können. Diese Umsetzung sollte durch die Verwendung des eLML-Frameworks erfolgen und anhand dessen praktischen Einsatzes untersucht werden, inwieweit sich das eLML Format für die Erstellung von E-Learning Inhalten eignet.

Weiterhin sollte das edb um eine Trainingsanwendung, speziell für den Bereich der Programmierung von Datenbanktriggern mit PL/SQL, erweitert werden. Die Anwendung sollte als Webanwendung, auf Basis von JSP, implementiert werden und zum Ziel haben, den Studierenden eine Möglichkeit zu schaffen ihre Kenntnisse im Bereich der Triggerprogrammierung interaktiv testen zu können.

In Kapitel 2 werden die Grundlagen erläutert, die zum besseren Verständnis dieser Arbeit benötigt werden. Hier wird das Thema PL/SQL, mit dem Schwerpunkt der Gestaltung von aktiven DBS mithilfe von Datenbanktriggern, behandelt sowie die Grundlagen der Entwicklung von Java basierten Webapplikationen mit JSP dargestellt. Besonders ausführlich werden hier die grundlegenden Konzepte des eLML Formats sowie dessen Entstehungshintergrund beschrieben.

Kapitel 3 beschreibt die Umsetzung der MS-Power-Point Folien in das eLML Format und verdeutlicht somit den praktischen Einsatz des eLML Frameworks von der Konfiguration der Entwicklungsumgebung bis hin zur Implementierung der Inhalte.

In Kapitel 4 wird die Konzeption und Implementierung der PL/SQL Trainingsanwendung dargestellt, wobei das Unterkapitel 4.3 die konzeptionellen Vorüberlegungen und verschiedene Alternativen von Lösungsmöglichkeiten aufzeigt und weiterhin die Auswahl der für die Implementierung gewählten Lösung begründet. Unter 4.4 wird die eigentliche Implementierung der Anwendungskomponenten detailliert beschrieben, hierbei werden Problematiken, die im Implementierungsprozess auftraten sowie deren Lösung nochmals gesondert herausgestellt.

Kapitel 5 stellt das abschließende Kapitel dieser Arbeit dar, hier werden die Ergebnisse und Erkenntnisse, die im Rahmen der Arbeit entstanden sind reflektiert und bewertet.

2 Grundlagen

2.1 PL/SQL

PL/SQL ist eine von Oracle entwickelte, prozedurale Erweiterung von SQL. Sie wurde entwickelt da SQL nicht hinreichend mächtig ist, um vollständige Anwendungssysteme zu beschreiben. Dies ist darauf begründet dass das Ergebnis einer SQL Anfrage immer eine Menge von Tupeln ist, wobei hier allerdings keine Möglichkeiten existieren die Tupel einzeln zu bearbeiten, Schleifen zu definieren oder Variablen zu definieren. Durch PL/SQL wird die Datenbanksprache um diese Programmierkonstrukte erweitert.

2.2 Aktive Datenbanken mit PL/SQL

2.2.1 Motivation

Konventionelle, passive Datenbanksysteme, in denen die Ausführung von Transaktionen ausschließlich durch Applikationsprogramme ausgelöst wird, haben einen entscheidenden Nachteil. Es besteht seitens des DBS keinerlei Möglichkeit bei einer Datenmanipulationen komplexere Integritätsprüfungen durchzuführen, die über die Möglichkeiten von check- oder foreign-key Bedingungen hinausgehen. Solche Prüfungen müssen hierbei von Seiten der Applikation erfolgen, die auf die Datenbank zugreift. Dies wirkt sich insbesondere dann nachteilig aus, wenn es mehrere unterschiedliche Anwendungen gibt, die auf dem gleichen Datenbestand arbeiten, da hierbei die entsprechende Anwendungslogik in jeder dieser Anwendungen implementiert und gewartet werden muss.

Aktive Datenbanken haben hier den Vorteil, dass sie auf Ereignisse von außen mit entsprechender Programmlogik reagieren können. Diese aktiven Komponenten werden als Datenbanktrigger oder einfach nur als Trigger bezeichnet und werden zentral in der Datenbank erstellt. Trigger sind spezielle Struktureinheiten und werden in Oracle Datenbanken in der Sprache PL/SQL implementiert, sie ist eine prozedurale Erweiterung von SQL und stellt einen fundamentalen Bestandteil von Oracle Datenbanken dar. Bei der Implementierung eines Triggers, kann der volle Sprachumfang von PL/SQL genutzt werden.

2.2.2 Anwendungsgebiete

Einige typische Anwendungsgebiete von Triggern sind unter Anderem:

Folgeverarbeitung:

Hierunter versteht man die Pflege von redundanten Daten innerhalb eines Datenmodells sowie die Berechnung abhängiger Datenfelder. Die Speicherung von redundanten Daten kann verschiedene Gründe haben, zum einen kann es in bestimmten Fällen zur Verbesserung der Performance führen, zum anderen auf historisch gewachsenen Datenmodellen beruhen. In den Bereich der Folgeverarbeitung fallen weiterhin auch Historisierungs- und Archivierungsfunktionalitäten, die bei der Änderung bestimmter Daten ausgeführt werden.

Konsistenzüberwachung:

Wie schon ansatzweise erwähnt, ist die Ausdruckskraft der CHECK-Constraints Bedingungen oft nicht ausreichend, um einen konsistenten Datenbestand sicherzustellen. Durch die Verwendung von Triggern, können wesentlich komplexere Bedingungen geprüft werden und auf Integritätsfehler kann individuell reagiert werden.

Benachrichtigungssysteme:

Durch Datenbanktrigger können im Fehlerfall entsprechende Fehlermeldungen mitprotokolliert oder direkt auf dem Bildschirm ausgegeben werden.

2.2.3 Bestandteile und Syntax eines Triggers

Trigger-Name

CREATE OR REPLACE TRIGGER Triggername

Trigger-Zeitpunkt

[BEFORE | AFTER }

Trigger-Ereignis

{ INSERT | UPDATE [OF [Spalte.[,Spalte]...] | DELETE ON Tabellenname }

Trigger-Typ

[FOR EACH ROW] befehlsorientiert, andernfalls zeilenorientiert.

Trigger-Restriktion

[WHEN-<Bedingung>

Trigger-Rumpf

<PL/SQL-Block>

Trigger-Name

Für den zu erstellenden Trigger muss ein eindeutiger Name vergeben werden, dieser kann frei gewählt werden, sollte jedoch erkennen lassen auf welche Tabelle er sich bezieht und welchen Zweck er erfüllt.

Trigger-Zeitpunkt

Für jeden Trigger muss ein Ausführungszeitpunkt definiert werden, er kann entweder vor oder nach dem auftretenden Ereignis ausgeführt werden.

Trigger-Ereignis

Der Trigger muss auf ein oder mehrere festgelegte Ereignisse reagieren, bei der Definition mehrerer Ereignisse kann innerhalb des Triggers abgefragt werden welches dieser Ereignisse eingetreten ist.

Trigger-Typ

Es existieren zwei Arten von Triggertypen, Row-Level Trigger werden für jede einzelne Zeile, die durch die auslösende DML-Anweisung verändert wurde, ausgeführt und Statement-Trigger, die jeweils nur einmal ausgeführt werden. In zeilenorientierten Triggern ist es möglich auf die Spaltenwerte vor und nach der auslösenden Aktion zuzugreifen, dies ist durch die Ausdrücke

:OLD.Spaltenname

:NEW.Spaltenname

möglich.

Trigger-Restiktion

Die Triggerdefinition kann zusätzlich eine optionale „WHEN“ Bedingung, ähnlich einer SELECT-Anweisung, enthalten.

2.3 MVC-Architektur

Das MVC-Architekturkonzept wurde ursprünglich für die Erstellung von Benutzeroberflächen mit Smalltalk-80 verwendet.[1] Das Konzept sieht die Trennung einer Anwendung in die Teilbereiche Datenmodell, Präsentation und Steuerung vor. Als eindeutige Bezeichnung für die Teilbereiche werden die Begriffe Model, View und Controller verwendet. In dieser Architektur wird die Anwendungslogik von der Präsentation und der Steuerungslogik getrennt implementiert, wodurch es einfacher möglich ist einzelne Teile wiederzuverwenden oder auszutauschen. Die folgende Abbildung zeigt das Modell sowie die Interaktionen der Komponenten miteinander.

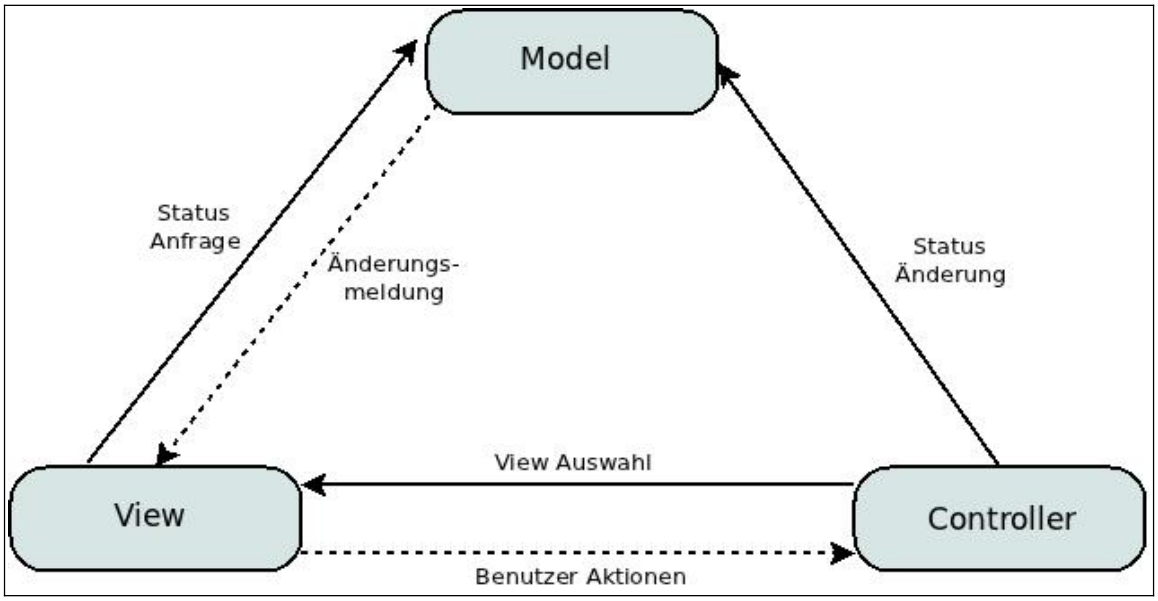


Abbildung 1: MVC- Architektur

Model

Das Modell repräsentiert die Geschäftslogik sowie die darzustellenden Daten. Bei Änderungen an relevanten Daten werden die im Modell registrierten Views informiert.

View

Die View kennt sowohl die Steuerung als auch das Modell, dessen Daten sie darstellt. Die Benutzeraktionen werden an die Steuerung weitergeleitet, in der die weitere Verarbeitung stattfindet. Die View wird vom Modell bei Datenänderungen benachrichtigt und kann die geänderten Daten abrufen.

---

[1] Vgl. [Gamma,1995] S.14

Controller

Die Controller Komponente nimmt die Benutzeraktionen entgegen, wertet sie aus und führt die entsprechenden Logiken des Modells aus. Weiterhin entscheidet sie welche View, zur Anzeige der Daten, verwendet werden soll.

Dieses Architekturmuster ist in der Praxis weit verbreitet und existiert mittlerweile in verschiedensten Abwandlungen.

2.4 Webanwendungen mit Servlets und JSP

2.4.1 Servlets

Servlets sind in Java geschriebene, eigenständige Programme, die auf einem dafür präparierten Server Anfragen von Clients über das HTTP Protokoll empfangen, verarbeiten und beantworten können. Zur Implementierung eines Servlets wird eine Klasse erzeugt, die von der Klasse *javax.servlet.http.HttpServlet* abgeleitet ist. In der erstellten Klasse werden nun die Methoden *doGet()* und *doPost()* implementiert, über diese werden die HTTP-GET und HTTP-POST Anfragen der Clients behandelt. Die Methoden besitzen jeweils einen Übergabeparameter vom Typ *HttpServletRequest,* zum Auslesen der Request Daten, sowie einen vom Typ *HttpServletResponse,* zum Generieren einer Antwort. Die Folgende Abbildung zeigt ein einfaches Beispiel einer Servlet Implementierung.

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloWorld extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                throws ServletException,IOException {

                response.setContentType ("text/html");
                PrintWriter out = response.getWriter();

                out.println("<HTML>");
                out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
                out.println("<BODY>");
                out.println("<BIG>Hello World</BIG>");
                out.println("</BODY></HTML>");
        }


}
```

Abbildung 2: Servlet Beispiel

2.4.2 Java Server Pages

Java Server Pages ist eine von Sun Microsystems entwickelte Programmiersprache zur Erzeugung von dynamisch generierten Web-Seiten. Die JSP Dateien bestehen aus reinem Text und enthalten im wesentlichen HTML mit eingebettetem Java Code, welche auf dem Server im Hintergrund zunächst in echten Java-Quellcode und weiterhin in Java Servlets umgewandelt werden. Der große Vorteil von JSP ist vor allem der, dass die HTML Ausgabe, anders als bei der Entwicklung mit Servlets, nicht innerhalb von Java Quellcode stattfindet und es damit einem Web-Entwickler möglich macht das Seitenlayout zu Gestalten, ohne Java Programmierkenntnisse zu besitzen. Somit kann, bei der Entwicklung von Java basierten Webanwendungen, eine saubere Trennung zwischen der Darstellung und der Applikationslogik vorgenommen werden. Die folgende Abbildung zeigt ein Beispiel einer JSP mit HTML Inhalt und eingebettetem Java Code.

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
</HEAD>
<BODY>
Hallo. Wir haben heute
<%= new java.util.Date() %>.
</BODY>
</HTML>
```

Abbildung 3: JSP Beispiel

Die Möglichkeit des Einbettens von Java-Quellcode in HTML macht die direkte Entwicklung von Servlets jedoch nicht überflüssig, diese beiden Techniken werden in der Praxis, innerhalb eines Entwicklungsprojekts, häufig gemeinsam eingesetzt. Für die Nutzung von JSP wird eine Umgebung benötigt, die einen JSP-Compiler und einen Servlet-Container zur Verfügung stellt, wie beispielsweise die Open-Source Ausführungsumgebung Apache Tomcat.

2.4.3 Java-Beans

Java-Beans sind einfache Komponenten, die aus einem anonymen Konstruktor und Attributen bestehen. Zu jedem Attribut existiert eine entsprechende get Methode zum lesen und eine set Methode zum setzen des Attributwerts.

2.4.4 Entwicklungsmodelle

Die JSP Spezifikation sieht zwei Vorgehensmodelle bei der Entwicklung von JSP basierten Web-Applikationen vor, die als Modell 1 und Modell 2 Architektur bezeichnet werden. Der Grundlegende Unterschied zwischen den Modellen ist der Ort, an dem die Verarbeitung stattfindet. In der Modell 1 Architektur nimmt eine JSP die Anfragen des Clients entgegen und liefert eine entsprechende Antwort zurück.

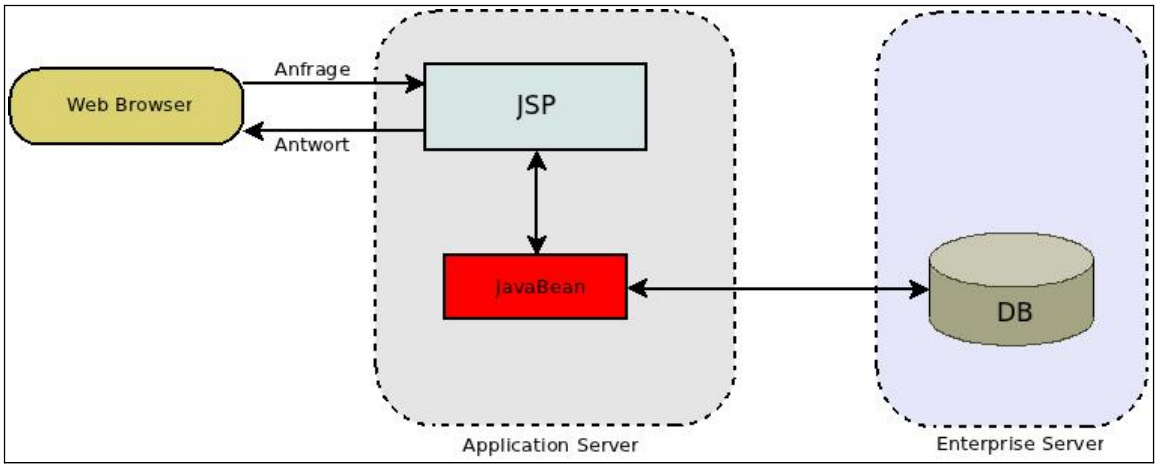


Abbildung 4: Modell 1 Architektur

Quelle: in Anlehnung an [Mahmoud, 2003]

Die Modell 2 Architektur verwendet sowohl JSPs als auch Servlets, wobei die JSPs ausschließlich für die Präsentation verwendet werden und die Anfrageverarbeitung in Servlets ausgelagert ist. Das Servlet stellt dabei eine Art Controller dar, es nimmt die Anfragen der Clients entgegen, erzeugt die für die JSP benötigten Beans und entscheidet an welche JSP die Anfrage weitergeleitet werden muss. Die JSP Seite empfängt die vom Servlet erzeugten Objekte und stellt deren Inhalte dar.

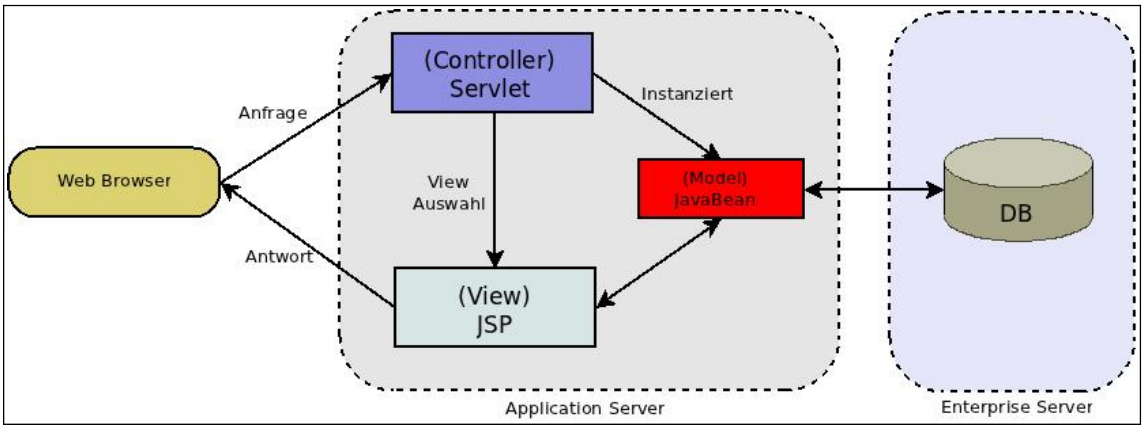


Abbildung 5: Modell 2 Architektur

Quelle: in Anlehnung an [Mahmoud, 2003]

In diesem Modell wird das MVC[2] Konzept als Grundlage verwendet und stellt somit eine Abwandlung dessen dar. Für die Applikationsentwicklung wird nach [Mahmoud, 2003] im allgemeinen die Modell 2 Architektur empfohlen, für sehr kleine Projekte kann jedoch auch die Modell 1 Architektur verwendet werden.

2.4.5 Gültigkeitsbereiche in Java-Web-Anwendungen

Da bei der Anwendungsentwicklung mit Servlets und JSPs unter Umständen Objekte erzeugt werden, die auch in weiteren Bereichen der Applikation verfügbar sein sollen, existieren bei der Entwicklung von Web-Anwendungen in JSP verschiedene Gültigkeitsbereiche zur Speicherung von benannten Objekten. Diese Bereiche können sowohl im Java Code der Servlets, als auch in den Skriptbereichen der JSPs verwendet werden. Insgesamt gibt es vier verschiedene dieser Gültigkeitsbereiche, die als *scopes* bezeichnet werden. Diese Bereiche gliedern sich nach der Dauer der Verfügbarkeit der beinhaltenden Daten und der Anzahl der Nutzer.

Name	Erklärung
application	Dieser Gültigkeitsbereich wird beim Start der Applikation initialisiert und steht bis zur Beendigung dieser zur Verfügung. Attribute die hier gespeichert werden können in allen Teilen der Applikation verwendet werden.
session	Dieser Bereich ist jeweils für eine Nutzersitzung gültig. Attribute die dort gespeichert werden bleiben so lange erhalten bis die Sitzung geschlossen wird und sind auch nur für die jeweilige Sitzung zugänglich.
request	Dieser Gültigkeitsbereich umfasst genau eine Anfrage eines Nutzers. Aufgrund einer möglichen Weiterleitung der Anfrage an weitere Servlets oder JSPs kann sich ein Request über mehrere JSPs oder Servlets erstrecken.
page	Dieser Gültigkeitsbereich ist nur in JSPs verfügbar und ist auch nur innerhalb genau einer JSP gültig. Bei einer Weiterleitung gehen Attribute dieses Gültigkeitsbereichs verloren.

Tabelle 1: Gültigkeitsbereiche in Java-Web-Anwendungen

[2] Vgl. Kapitel 2.3

2.5 ELML

2.5.1 Beschreibung

ELML ist ein Open-Source XML- Framework zur Erstellung von Strukturierten Lerneinheiten im XML Format. Es besteht grundlegend aus XML-Schema-Dateien, durch die die Struktur definiert wird, und weiterhin aus XSLT-Dateien für die Transformation des XML-Inhalts in verschiedene Ausgabeformate. Unterstützt werden hier zur Zeit die folgenden Ausgabeformate: HTML, PDF, ePub, SCORM, IMS, LATEX, DocBook sowie das ODF Format, weitere sind bereits in Planung. Die in der Praxis häufigst genutzten Ausgabeformate sind das HTML und das PDF Format.

2.5.2 Entstehung

Der Ursprung von eLML ist zurückzuführen auf das Projekt GITTA[3], dieses Projekt wurde durch das SVC[4] initiiert. Aufgabe innerhalb dieses Projekts war es einheitliche und doch flexible Module und Lektionen für den GIST[5] Unterricht zu erstellen.

Als das Projekt im Jahre 2001 gestartet wurde, bot keines der zu dieser Zeit erhältlichen, kommerziellen E-Learning Produkte die Möglichkeiten, die für dieses Projekt gewünscht waren. Vor allem die gewünschte Möglichkeit zur Übernahme von Inhalten in andere Systeme wurde von keiner vorhandenen E-Learning-Software unterstützt.

Die einzige Lösung bestand nun darin eine Eigenentwicklung durchzuführen. Diese Lösung sah vor, die Inhalte im XML Format zu speichern und sie durch Nutzung freier Software(Apache-Cocoon[6] wurde dafür verwendet) zur Verfügung zu stellen. Zur einheitlichen Strukturierung der Lektionen wurde auf Basis eines pädagogischen Konzepts zunächst ein DTD entwickelt, welches dann im späteren Verlauf in ein XML-Schema überführt wurde. Nach fast drei Jahren der Anwendung und stetiger Weiterentwicklung der GITTA Struktur, wurde die Entscheidung getroffen, das XML Schema und die entsprechenden XSL Dateien ,zur Erstellung von HTML (XSLT) und PDF (XSL-FO), der Öffentlichkeit zur freien Verfügung zu stellen. Das veröffentliche XML Schema, das auf dem GITTA DTD

[3] Geographic Information Technology Training Alliance
[4] Swiss Virtual Campus
[5] Geographic Information Science and Technology
[6] Apache Cocoon ist ein XML-Publishing-System, das die Trennung von Inhalt und Darstellung konsequent umsetzt. Weitere Informationen unter: http://cocoon.apache.org

basierte und in den letzten drei Jahren erfolgreich eingesetzt wurde, wurde mit entsprechenden XSL Dateien von Grund auf neu erstellt, wobei die Beeinträchtigungen und Bugs, die noch aus dem ursprünglichen DTD resultierten, behoben wurden. Das Resultat daraus wurde dann eLML genannt.

2.5.3 Pädagogisches Konzept

Die didaktische Struktur von ELML basiert auf dem pädagogischen Konzept ECLASS [Gerson,2000], wobei diese Struktur für die Verwendung in eLML entsprechend angepasst wurde. ECLASS steht für die Anfangsbuchstaben der einzelnen „unit“ Elemente: „entry“, „clarify“, „look“, „act“, „self-assessment“ und „summary“. Das „entry“ Element soll einleitende Informationen zu dem aktuellen „unit“ enthalten um zu verdeutlichen welche Thematik hier behandelt wird, während unter „clarify“ die eigentlichen Erklärungen folgen, somit stellt das „clarify“ Element den Kern eines jeden „units“ dar. Im „look“ Element folgen nun weitere Verdeutlichungen zu den vorangegangen Erklärungen anhand von Beispielen in Form von Bildern oder anderer multimedialer Inhalte. Das „act“ Element soll dem Leser die Möglichkeit geben, selbst aktiv zu werden und Anregungen geben das Erlernte anzuwenden, wobei das „self-assess“ Element dafür vorgesehen ist spezielle Übungsaufgaben, beispielsweise in Form von Multiple Choice Aufgaben, zu definieren. Diese beiden Elemente stehen somit in engem Bezug zueinander und können als eine Einheit betrachtet werden. Das abschließende Element „summary“ soll die in diesem „unit“ enthaltenen Inhalte und Lernziele nochmals zusammenfassend darstellen. Durch diese Struktur erhält der Ersteller von E-Learning Inhalten eine Art Leitfaden zu deren Strukturierung, wobei sie ihm jedoch einige Freiheiten der individuellen Gestaltung offen lässt. So sind die Elemente „clarify“, „look“ und „act“ so definiert, dass sie in beliebiger Reihenfolge und Anzahl verwendet werden können oder, wie beispielsweise das Element „self-assessment“, das sowohl auf „lesson“ als auch auf „unit“ Ebene eingesetzt werden kann.

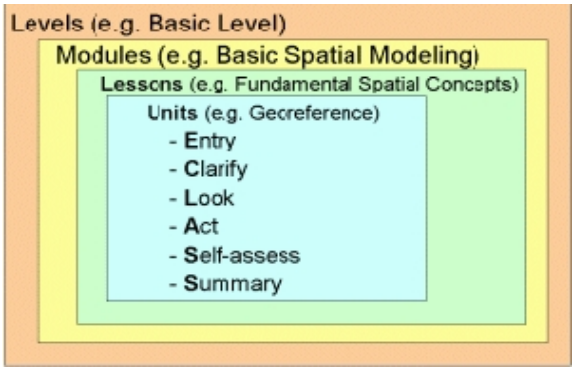


Abbildung 6: ECLASS Konzept

Quelle:[Bleisch, Fisler, 2010], S.8

2.5.4 Struktureller Aufbau der Lektionen

Die eLML Lektionen sind in sich abgeschlossene Einheiten. Die eLML Struktur beschreibt wie eine Lektion aufgebaut sein sollte, das heißt welche Elemente zwingend vorgegeben sind und welche optional verwendet werden können.

Die Strukturierung der eLML Lektionen findet auf drei Ebenen statt, im einzelnen sind das die „lesson“, „unit“ und „learningobjekt“ Ebene. Durch das „lesson“ Element wird die Lektion eingeleitet, auf dieser Ebene können nun, durch Verwendung des „entry“ Elements, einleitende Informationen zu der Lektion angegeben werden sowie weiterhin, durch das „goals“ Element, die Lernziele der Lektion definiert werden. Die Struktur schreibt hier die Verwendung von mindestens einem dieser beiden Elemente vor, die Verwendung beider Elemente ist selbstverständlich ebenfalls möglich. Weiterhin muss jede Lektion mindestens ein „unit“ Element enthalten, das „unit“ Element stellt die nächste Strukturebene dar und teilt die Lektion in einzelne Kapitel auf. Auf der „unit“ Ebene können wiederum, durch die Elemente „entry“ und „goals“, einleitende Information sowie die Lernziele des betreffenden Abschnitts angegeben werden, sind hier jedoch, anders als im „lesson“ Element, optional. Einzig die Definition von mindestens einem „learningobjects“ ist auf dieser Ebene zwingend vorgegeben. Das „learningobject“ Element stellt somit die dritte und letzte Gliederungsebene des eLML Formats dar, innerhalb dieser befinden sich nun die eigentlichen Inhalte. In der folgenden Abbildung ist die Struktur des eLML Formats graphisch dargestellt.

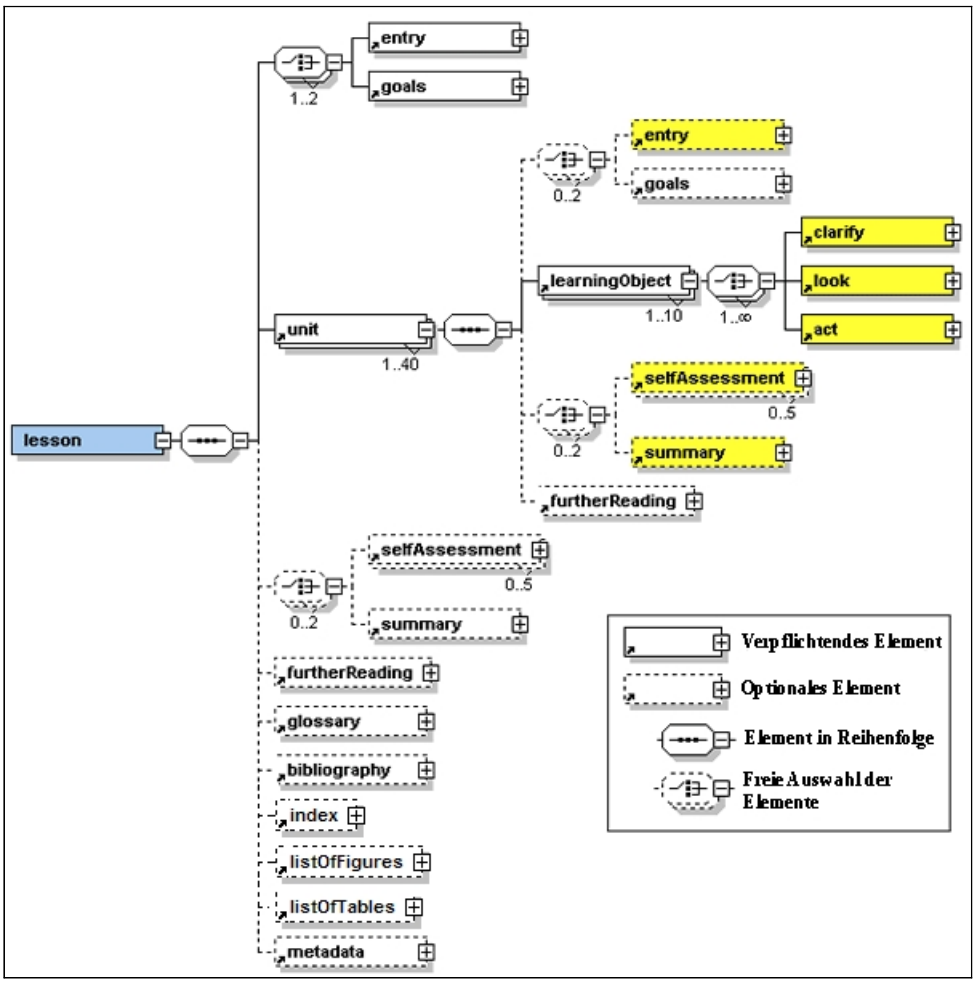


Abbildung 7:eLMLStruktur

Quelle:in Anlehnung an [Bleisch, Fisler, 2010], S.9

2.5.5 Inhaltselemente

Zur Erstellung der Inhalte innerhalb der Strukturelemente sind in eLML eine Reihe weiterer Elementen definiert. Sie ermöglichen das hinzufügen von Texinhalten, Bildern und weiteren multimedialen Inhalten innerhalb der Lektionen. Durch die Elemente „column“, „table“ und „list“ können die Inhalte strukturiert werden, „box“, „paragraph“ sowie „popup“ definieren unterschiedliche Textabschnitte. Innerhalb von Textabschnitten besteht die Möglichkeit mit „newline“ explizite Zeilenumbrüche zu definieren, bestimmte Teile mit „formatted“ speziell hervorzuheben und mit „citiation“ fremde Quellen zu zitieren. Durch das „term“ Element kann auf Glossareinträge referenziert werden und mit

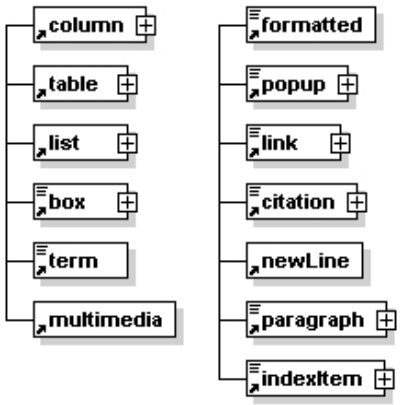


Abbildung 8: eLML Elemente

„indexItem“ können Wörter gekennzeichnet werden, aus denen dann später automatisch ein Index erstellt wird. Das „link“ Element erlaubt es Verlinkungen zu Abschnitten innerhalb der Lektion, zu fremden Lektionen sowie zu externen Webseiten zu erstellen. Mit Hilfe von „multimedia“ lassen sich multimediale Inhalte jeglicher Art einfügen, beispielsweise Bilder, Flash-Animationen, Audio oder gar Java-Applets. Multimediale Inhalte, die nicht explizit unterstützt werden, lassen sich dennoch einbinden, wenn auch nicht ganz so komfortabel. Dazu wird das Attribut „type“ auf „div“ gesetzt, wodurch innerhalb des Elements HTML Code verwendet werden kann.

2.5.6 Definition von Übungsaufgaben

Das eLML Format stellt Möglichkeiten zur Definition von Übungsaufgaben unterschiedlicher Art zur Verfügung. Diese werden über das „selfcheck“ Element eingeleitet, welches auf „lesson“ sowie auf „unit“ Ebene innerhalb des Elements „selfassessment“ verwendet werden kann, während es innerhalb der „learningobject“ Ebene innerhalb des „act“ Elements verfügbar ist. Es können zum einen Single und Multiple Choice Aufgaben erstellt werden. Der Unterschied zwischen diesen beiden Aufgabentypen besteht darin, dass bei Single Choice Aufgaben genau eine richtige Lösung ausgewählt werden muss, während es bei Multiple Choice Aufgaben mehrere richtige Lösungen gibt. Zum anderen gibt es die Möglichkeit Lückentext Aufgaben zu verfassen, die durch Texteingaben des Anwenders vervollständigt werden müssen. Die folgenden Abbildungen zeigen jeweils ein Beispiel dieser Aufgabentypen.

Single Choice

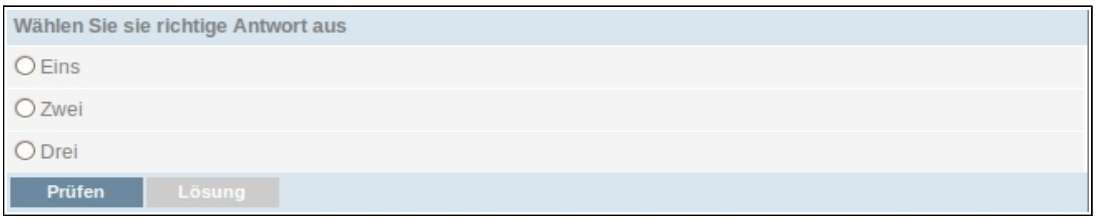


Abbildung 9: eLML Single Choice Aufgabe

Multiple Choice

Abbildung 10: eLML Multiple Choice Aufgabe

Lückentext

Abbildung 11: eLML Lückentext Aufgabe

Durch Anklicken des „Prüfen“ Buttons wird dem Anwender angezeigt, ob seine angegebene Lösung richtig war. Weiterhin ist es in beiden Aufgabentypen möglich die korrekte Lösung zu hinterlegen, diese kann durch Klicken auf den „Lösung“ Button zur Anzeige gebracht werden.

2.5.7 Festlegen des Darstellungs-Layouts

Die Inhalte der eLML Lektionen werden durch die Verwendung der beschriebenen Elemente innerhalb eines XML Dokuments definiert, an dieser Stelle befinden sich jedoch keinerlei Angaben bezüglich der späteren Darstellung der Inhaltselemente. Der Grund dafür ist, dass in eLML Inhalt und Darstellungsinformation strikt von einander getrennt werden. Die Darstellung der Lektionen wird durch externe XSL/XSLT und CSS Dateien gesteuert, wodurch man ein einheitliches Layout für alle erstellten Einheiten erhält, ohne Seitenlange Darstellungsvorschriften verfassen zu müssen. Die XSLT Dateien definieren hierbei das Layout der Lektionen, beispielsweise wo sich die Navigation befinden soll, an welcher Stelle der Inhalt dargestellt wird sowie Kopf- und Fußleisten. Die folgende Abbildung zeigt ein

einfaches Beispiel eines solchen XSLT Templates, hier wird ein zweispaltiges Layout definiert, das in der linken Spalte die Navigation und in der rechten den Inhalt darstellt.


```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:elml="http://www.elml.ch" versi
<xsl:import href="../../../core/presentation/online/elml.xsl"></xsl:import>
<xsl:param name="layout" select="'online'"></xsl:param>
<xsl:param name="chapter_numeration" select="'no'"></xsl:param>
<xsl:param name="css_framework" select="'yaml'"></xsl:param>
<xsl:template name="elml:LayoutBody">
<body>
<div id="page_margins">
<div id="page">
<div id="header">
<div id="hcol_alone">
<div id="hcol1_content">
 </div></div></div><div id="main">
<div id="col1">
<div id="col1_content" class="nav">
        <xsl:call-template name="elml:navigation"></xsl:call-template>
</div></div><div id="col3">
<div id="col3_content">
        <xsl:call-template name="elml:LayoutBodyContent"></xsl:call-template>
</div></div></div><div id="footer">
<div id="fcol_alone">
<div id="fcol1_content">
</div></div></div></div></div></body></xsl:template></xsl:stylesheet>
```

Abbildung 12: Beispiel: XSLT Layout Template

Die CSS Dateien enthalten Informationen darüber, wie die Verwendeten Inhaltselemente dargestellt werden bezüglich Schriftart und Größe, Farbgebung sowie weiterer elementspezifischer Attribute wie beispielsweise der Rahmenstärke bei Verwendung des „box“ Elements. Die Abbildung stellt ein Auszug einer solchen CSS Datei dar, in der die Darstellung verschiedener eLML Elemente definiert wird.

```
sub, sup /* Used in the eLML element "formatted" */ { font-size: 0.6em }.element {
 font-family: "Courier New", Courier, Monaco, monospace;
 font-weight: bold;
 text-align: left;}

.box, .popup /* The eLML box or popup element. */ { background-color: #BADE17;
 background-position: 0 100%;
 padding: 0 1em 1em ;
 border: solid 2px #747566}

.fancylooking /* The eLML box example class. */ { background-color: red;
 padding: 1em ;
 border: solid 2px black;
 margin-bottom:2em;
 width: 100px;}

.tutor/* Paragraphs or parts only visible to tutors will be displayed like this. */ { color: red;
 font-weight: bold;
 background-color: yellow }

.multimedia_paragraph_center/* for multimedia objects that are used as paragraph and have align=center */ { tex
 width: 100%;
 float: none;
 padding: 1em;
 display: block;
 }
.multimedia_paragraph_right/* for multimedia objects that are used as paragraph and have align=right */ { text-
 float: none;
 padding: 1em 0 1em 1em;
 display: block;
 }
```

Abbildung 13: Beispiel: CSS Layout

2.5.8 Transformation in ein Ausgabeformat

Nachdem die Inhalte in die XML Datei eingepflegt wurden und ein Layout Template erstellt wurde, kann nun die Transformation in eines der unterstützen Ausgabeformate durchgeführt werden. Für die Transformation von eLML Lektionen wird ein XSLT 2.0 fähiger XSLT-Prozessor, wie beispielsweise SAXON ab der Version 8, benötigt. In der nachfolgenden Abbildung ist Transformationsprozess von XML in das HTML Ausgabeformat beispielhaft dargestellt.

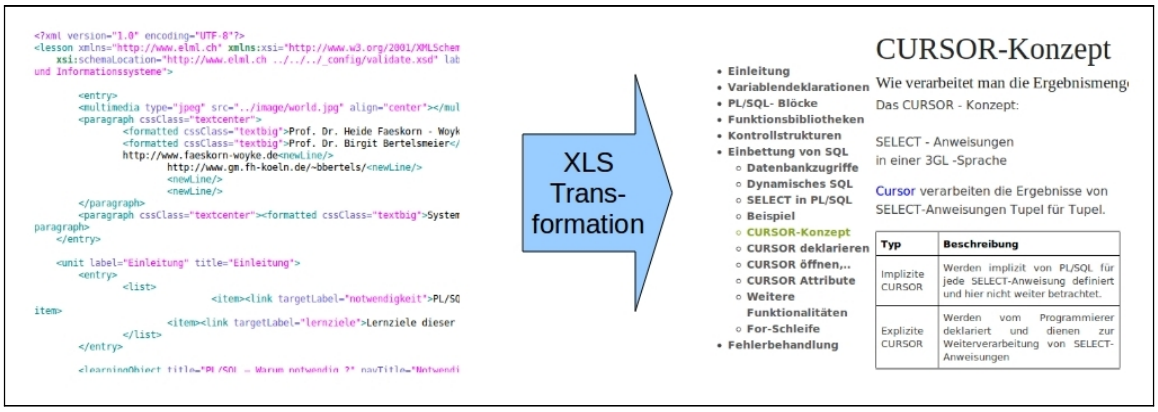


Abbildung 14: XSL-Transformation Beispiel

2.5.9 Verwendung des eLML Formats

Die Erstellung von eLML Lerneinheiten findet innerhalb eines XML-Editors statt und setzt zumindest grundlegende Kenntnisse im Bereich HTML, XML und XSLT voraus. Mittlerweile existieren zwar diverse WYSIWYG Tools zur Bearbeitung von eLML Lektionen, diese bieten jedoch nur eingeschränkte Funktionalität. Mit ihnen können bereits bestehende Einheiten zwar inhaltlich bearbeitet werden, strukturelle Änderungen sowie das Anlegen neuer Lektionen sind hiermit jedoch nicht möglich. Jemand, der bereits über entsprechende Vorkenntnisse verfügt, findet sich hier schnell zurecht und kann bereits nach wenigen Stunden des Selbststudiums mit der Erstellung einer ersten Lektion beginnen.

3 eLML Lektionen

3.1 Anforderungen

Innerhalb dieser Arbeit wurden bereits die theoretischen Aspekte von eLML behandelt. Weiterhin soll aber auch detailliert beschrieben werden, wie das eLML Format in der Praxis eingesetzt wird. Im Folgenden wird dies anhand eines Beispielprojekts verdeutlicht. In dem Beispielprojekt werden zwei vorlesungsbegleitende Foliensätze des Fachs DuI mit dem Themenschwerpunkt PL/SQL, vorliegend im Power-Point Format, ins eLML Format übertragen.

Die erste Einheit behandelt das Thema PL/SQL Programmierung im Allgemeinen, die zweite beschäftigt sich speziell mit der Entwicklung von Datenbanktriggern unter Verwendung von PL/SQL.

Aus den eLML Lektionen soll im Anschluss eine Ausgabe im HTML Ausgabeformat generiert werden. Das Layout der Ausgabe soll zweispaltig gestaltet sein, in der linken Spalte die Navigation und in der Rechten den Inhalt darstellen.

3.2 Verwendete Produkte

3.2.1 Eclipse

Zur Erstellung des XML Quellcodes sowie dessen Transformation durch XSLT wurde die Entwicklungsumgebung Eclipse in der Version 3.6.1(Helios) eingesetzt. Eclipse wurde deshalb gewählt, da es bereits standardmäßig Funktionalitäten enthält, XML Dokumente komfortabel zu bearbeiten. Zudem ist es kostenlos verfügbar.

3.2.1.1 Orangevolt Plugin

Für die XSL-Transformation wird ein XSLT 2.0 Prozessor benötigt. Da weder in Eclipse, noch im JDK ein solcher enthalten ist, kommt hier das Eclipse-Plugin OrangeVolt in der Version 1.0.7 zum Einsatz. Das Plugin integriert den XSLT Prozessor SAXON 8.9b, der die XSLT 2.0 Spezifikation in vollem Umfang unterstützt, sowie Funktionalitäten zur Durchführung der XSL-Transformation bietet.

3.2.2 ELML

Das eLML XML Framework wurde in Version 6.0 eingesetzt. Dieses Framework beinhaltet Funktionalitäten, den nach definierten Vorgaben erstellten XML Inhalt, in verschiedenste Präsentationsformate zu Transformieren. Ebenfalls enthalten ist hier der „Template Builder“ zur Generierung von XHTML Ausgabe Templates.

3.2.3 YAML

Das (X)HTML/CSS Framework YAML wurde in der Version 3.3 verwendet. Benötigt wurde es aus dem Grunde, da der „Template Builder“ des eLML Frameworks die YAML Bibliotheken als Grundlage voraussetzt.

3.3 Umsetzung

3.3.1 Konfiguration der Entwicklungsumgebung

Zur Verwendung von eLML musste zunächst das „core“ Verzeichnis, des eLML Frameworks, dem aktuellen Eclipse Workspace als Projekt hinzugefügt werden. Es enthält im Unterverzeichnis „presentation“ XSLT Skripts für alle zur Zeit unterstützten Ausgabeformate. Im Rahmen dieses Projekts wurde das „online“ Ausgabeformat verwendet, durch welches (X)HTML Dateien generiert werden. Im Ordner „structure“ befinden sich die XML Schemata, die zur Validierung benötigt werden.

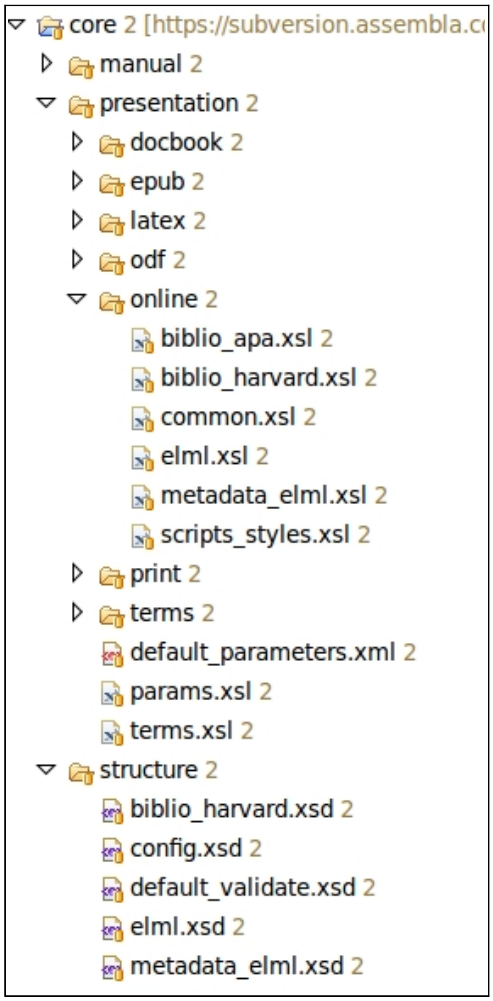


Abbildung 15: eLML Framework Verzeichnisstruktur

3.3.2 Erstellung eines Ausgabe Templates

Das eLML-Template, das zur Transformation der XML Inhalte nach XHTML benötigt wird, wurde mit dem „Template Builder“ erstellt. Hierbei handelt es sich um eine webbasierte Anwendung, die speziell für die Generierung von eLML Templates entwickelt wurde. Als Output des „Template Builders“ wird zum einen ein XSL Transformations Skript erstellt, zum anderen eine CSS Datei.
Das Layout ist zweispaltig, die linke Spalte enthält die Navigation und in der Rechten wird der Inhalt angezeigt. Auf einen Header sowie einen Footer wurde in diesem Falle bewusst verzichtet, da die Lektionen innerhalb des edb in einem Iframe dargestellt werden.

3.3.3 Erzeugung des Projekts

Für die Erzeugung eines neuen eLML Projekts ist im eLML Framework bereits ein leeres Projekt enthalten, welches als Basis verwendet und ebenfalls in Eclipse eingebunden wurde. Der Name des eingebunden Projekts wurde von der Standardbezeichnung „MyProject“ in „DuI“ geändert. Damit innerhalb des Projekts das zuvor erstellte Ausgabe Template verwendet werden konnte, musste dem Unterverzeichnis „_templates“ zum einen das Template selbst, zum anderen der Ordner „yaml“ aus dem YAML Toolkit hinzugefügt werden, da das Template auf die YAML Bibliotheken referenziert. Das Verzeichnis „MyLesson“ repräsentiert eine leere Basislektion, hiervon wurden auf gleicher Ebene zwei Kopien angelegt, eine davon wurde umbenannt in „plsql“, die andere in „aktivedbs“. In beiden somit erzeugten Lektionen wurde die Bezeichnung des darunterliegenden Ordners „en“ nach „de“ geändert, wodurch die Projektsprache von englisch auf deutsch angepasst wurde. Innerhalb der „de“ Verzeichnisse befindet sich in dem Ordner „text“ die XML Datei, die den eigentlichen Inhalt darstellt und ebenfalls auf die Benennung der jeweiligen Lektion angepasst wurde. In der nachfolgenden Abbildung ist die Struktur des erstellten eLML Projekts veranschaulicht.

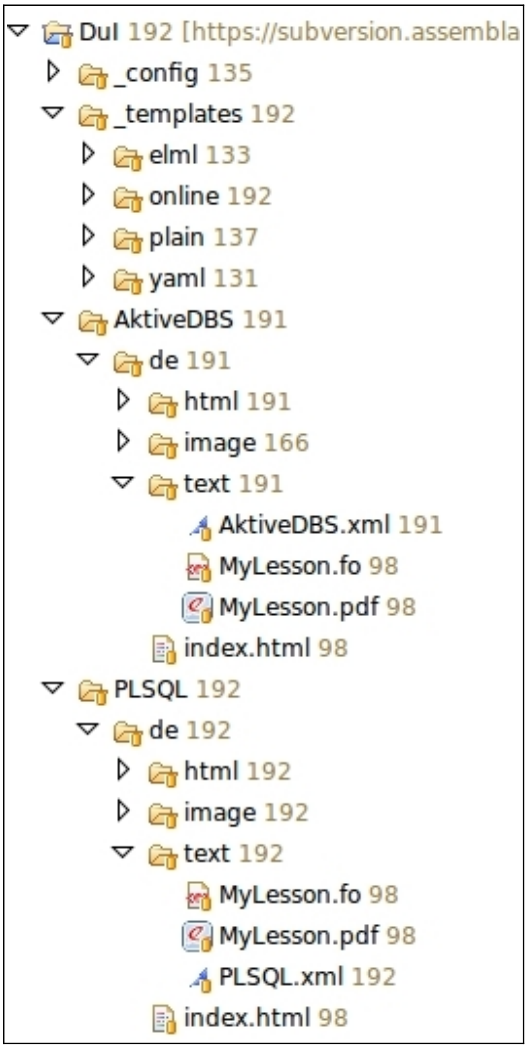


Abbildung 16: eLML Projektstruktur

3.3.4 Strukturierung der Lektionen

Nachdem die nötigen Vorarbeiten zur Implementierung der Lektionen abgeschlossen sind, müssen die Lektionen in eine sinnvolle Struktur gebracht werden. Strukturiert wird auf drei Ebenen, an oberster Stelle steht die „lesson“ Ebene, welche die Lektion an sich darstellt. Darunter befinden sich die „units“, wodurch die Lektion in verschiedene Themenbereiche aufgeteilt wird. In jedem dieser „unit“ Elemente befinden sich mehrere „learningobjects“, diese beinhalten den zu vermittelnden Lerninhalt.

3.3.4.1 Lektion: PL/SQL

Diese Lektion beschäftigt sich mit der Sprache PL/SQL im allgemeinen, hier werden die verschiedenen Sprachelemente aufgezeigt und erläutert sowie in welchem Kontext sie verwendet werden. Die Lektion beginnt mit dem „unit“ Einleitung, in dem die Notwendigkeit

von PL/SQL herausgestellt wird sowie die Lernziele dieser Lektion aufgezeigt werden. Darauf folgt das „unit“ Variablendeklaration, in den darin enthaltenen „learningobjects“ werden die in PL/SQL enthaltenen Datentypen aufgezeigt, wie mithilfe der vordefinierten Typen eigene zusammengesetzte Datentypen definiert werden können und wie Vektoren deklariert und verwendet werden. Unter dem „unit“ Blöcke werden die unterschiedlichen Blockstrukturen erläutert, die in PL/SQL möglich sind und für welche Einsatzzwecke sie sich jeweils eignen. Das „unit“ Bibliotheken befasst sich mit dem Thema PL/SQL Packages, in den enthaltenen „learningobjects“ wird beschreiben wie Packages aufgebaut sind, welchen Zweck sie erfüllen und wie sie verwendet werden, in diesem Zuge werden außerdem die objektorientierten Ansätze von PL/SQL herausgestellt. Das folgende Kapitel widmet sich der Beschreibung der zur Verfügung stehenden Kontrollstrukturen, Möglichkeiten der Programmverzweigung sowie verschiedener Schleifenkonstrukten. Im „unit“ Einbettung wird dargestellt wie SQL Anweisungen innerhalb von PL/SQL Code eingebettet werden und wie dynamisch generierter SQL Code zur Ausführung gebracht wird, weiterhin wird hier schrittweise das CURSOR- Konzept erläutert und anhand von Beispielen erklärt wie durch Verwendung von Cursorn Ergebnismengen durchlaufen werden können. Das abschließende „unit“ dieser Lektion behandelt das Thema Fehlerbehandlung mit Hilfe von Exceptions. Hier werden die unterschiedlichen Exceptiontypen und alle bereits vordefinierten Exceptions aufgelistet und weiterhin dargestellt wie Exceptions aus dem PL/SQL Code heraus ausgelöst werden.

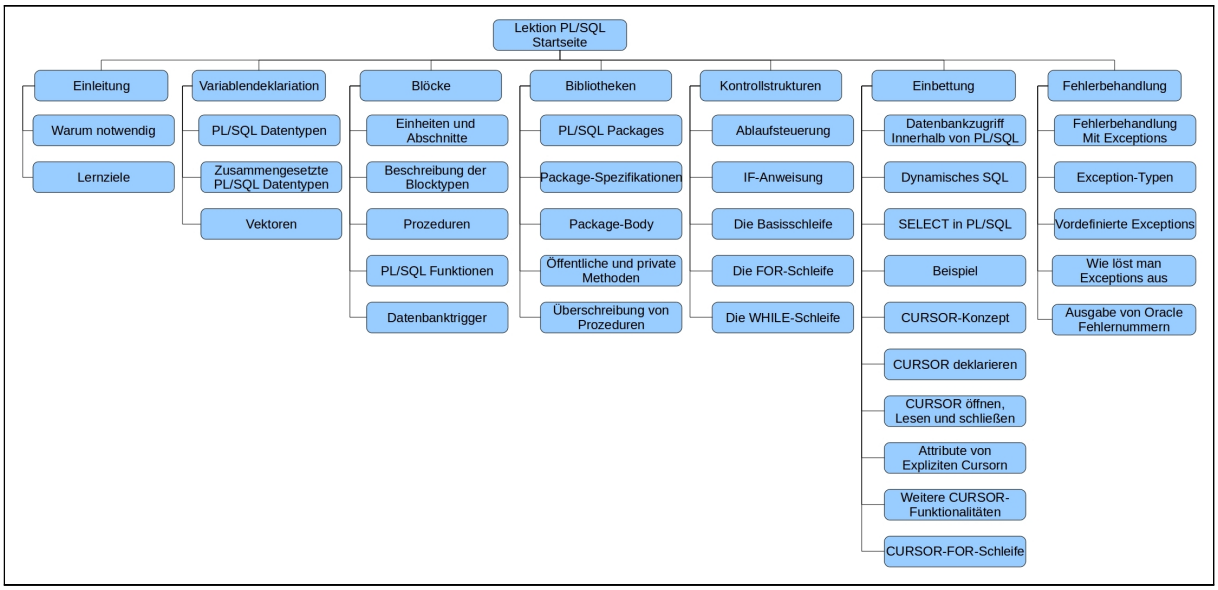


Abbildung 17: Struktur der Einheit:PL/SQL

3.3.4.2 Lektion: Aktive DBS

Die Lektion „Aktive DBS“ stellt eine weiterführende Einheit zum Thema PL/SQL dar und beschäftigt sich speziell mit der Programmierung von Datenbanktriggern in PL/SQL. Das erste „unit“ stellt zunächst die grundlegende Motivation für die Verwendung von Datenbanktriggern dar und behandelt im Anschluss die zugrunde liegenden ECMA- Regeln. Das darauf folgende „unit“ befasst sich mit der Trigger Syntax, hier wird beschrieben wie Trigger definiert werden und aus welchen Bestandteilen diese bestehen. Weiterführend wird dargestellt welche Trigger-Typen es gibt und auf welche Art von Ereignissen ein Trigger reagieren kann. Das „unit“ Ausführungsmodelle zeigt die unterschiedlichen Ausführungsmodelle von Datenbanktriggern mit entsprechenden Beispielen dazu. Im „unit“ Triggerfallen wird beschrieben welche Probleme bei der Entwicklung von Triggern auftreten können, insbesondere wird hier das Mutating-Table Problem behandelt. Das abschließende „unit“ beschreibt wie man über SQL-Statements, die in einer Datenbank vorhandenen Prozeduren und Trigger abfragen kann.

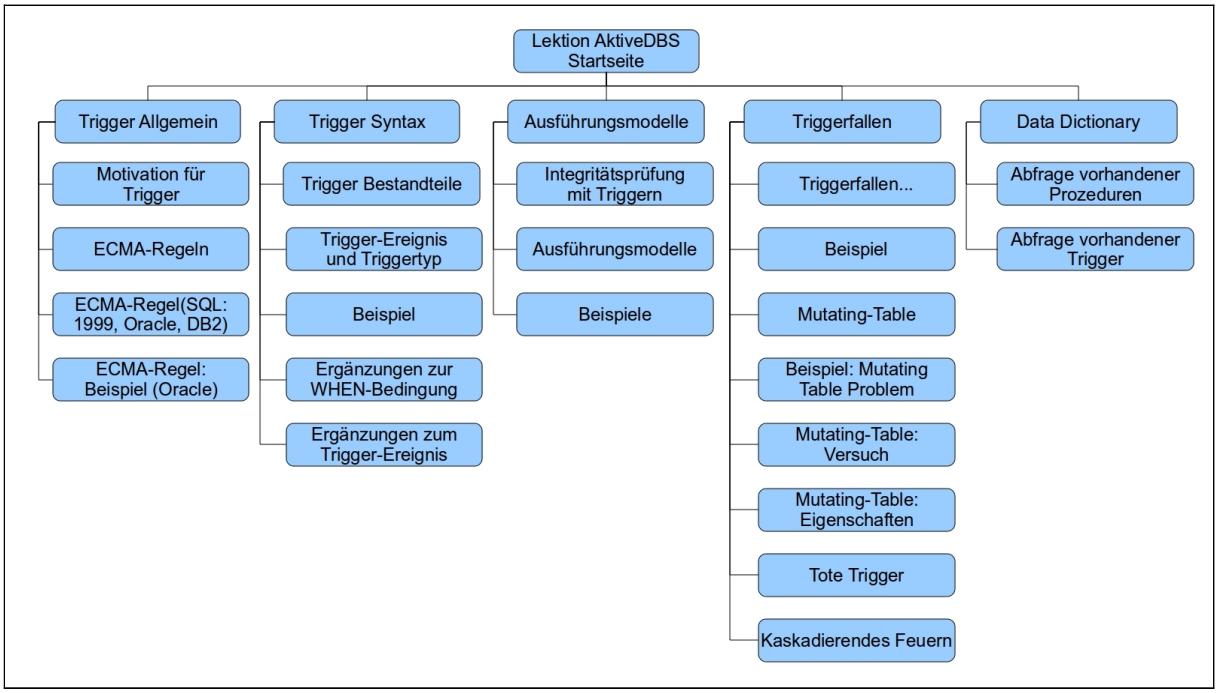


Abbildung 18: Struktur der Einheit:Aktive DBS

3.3.5 Implementierung der Inhalte

Die Inhalte aus den bestehenden Folien konnten größtenteils problemlos ins eLML Format übernommen werden. Gewöhnliche Textinhalte wurden mit Hilfe des „paragraph“ Elements eingepflegt, bei Code Beispielen wurde innerhalb des „paragraph“ Elements das Element

„formatted“, zur farblichen Kennzeichnung von Schlüsselwörtern, verwendet. Listen sowie die Übernahme von Tabellen waren ebenfalls durch die Elemente „list“ und „table“ möglich. Die nachfolgende Abbildung zeigt einen beispielhaften Vergleich der Darstellung des gleichen Inhalts zwischen der der Power-Point Folien und der Darstellung im eLML Format.

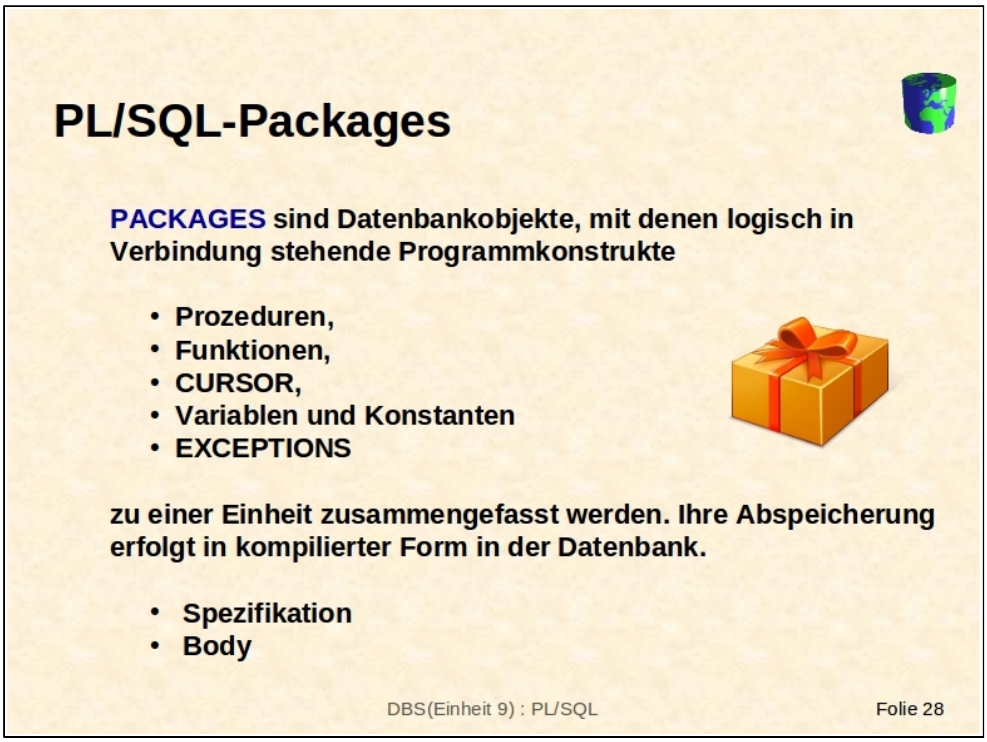


Abbildung 19: Darstellung Power-Point

Abbildung 20: Darstellung eLML Format

Problematisch bei der Umsetzung waren jedoch Inhalte, die wie folgt gestaltet waren.

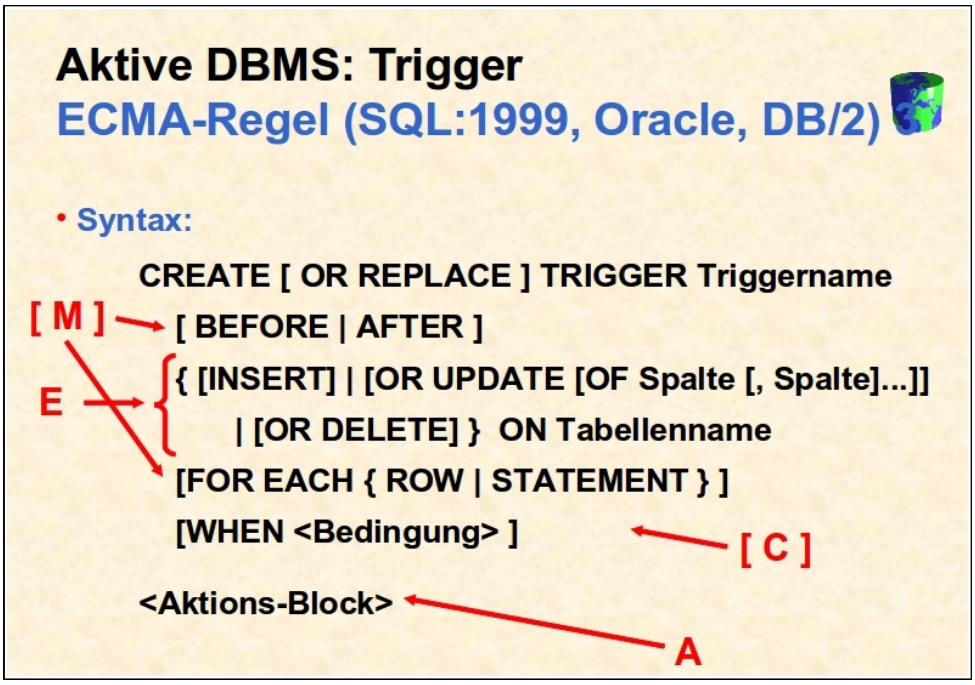


Abbildung 21: Beispiel einer problematischen Power-Point Folie

Problematisch hierbei ist, dass hier Textinhalte mit grafischem Inhalt vermischt sind, das eLML Format bietet keine Möglichkeit einer solchen Darstellung an.

Dieses Problem wurde dadurch gelöst, dass sowohl der Text als auch die grafischen Elemente zusammen in einer Bilddatei gespeichert und in eLML eingebunden wurde, dadurch bleibt die Darstellung von Inhalten dieser Art in den Lektionen originalgetreu. Die Nachteile dieses Verfahrens äußern sich jedoch dann, wenn sich nachträglich noch Änderungen innerhalb einer dieser Texte ergeben, da hierzu die komplette Bilddatei ausgetauscht werden muss.

3.3.6 XSL Transformation

Für die Transformation der erstellten XML Datei in ein (X)HTML Ausgabeformat wird ein entsprechendes XSL Transformations- Skript sowie ein geeigneter XSLT- Prozessor benötigt. Als XSLT Skript wird hier das aus dem, bereits zuvor erstellten, Ausgabe Template verwendet. Mit Hilfe des Eclipse Plugins „OrangeVolt“ wird die Transformation durchgeführt, als Parameter müssen hier lediglich die betreffende XML Datei sowie das XSLT Dokument aus dem Template angegeben werden. In der folgenden Abbildung ist der Transformationsprozess im Detail dargestellt.

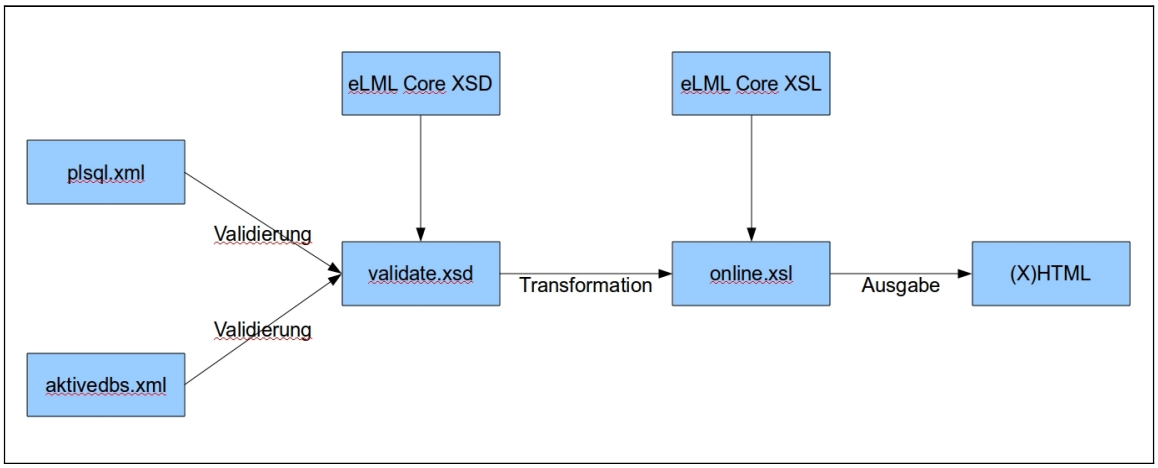


Abbildung 22: XSL Transformationsprozess

4 PL/SQL Trainingsanwendung

4.1 Anforderungen an die Applikation

Das edb der FH-Köln sollte um PL/SQL Trainingsanwendung mit dem Schwerpunkt Triggerprogrammierung erweitert werden. Dem Anwender sollen hierbei zu den verschiedenen Anwendungsbereichen von Triggern entsprechende Aufgaben gestellt werden, welche durch die Eingabe von PL/SQL Code beantwortet werden müssen. Die eingegebene Lösung soll auf ihre Korrektheit geprüft und das Ergebnis dem Anwender präsentiert werden. Die Aufgabe soll dann als richtig gelöst gelten, wenn nach Auslösung des Triggers der Tabelleninhalt einer für die entsprechende Aufgabe festgelegten Tabelle mit dem Inhalt einer Musterlösung übereinstimmt, was zur Folge hat, dass es mitunter mehrere richtige Lösungen zu einer Aufgabe geben kann. Die Anwendung soll, wie die bereits bestehenden Trainingsapplikationen, in JSP entwickelt werden und innerhalb einer Tomcat 6 Ausführungsumgebung lauffähig sein.

4.2 Verwendete Tools und Techniken

4.2.1 Eclipse

Zur Erstellung der JSP Anwendung wurde die Entwicklungsumgebung Eclipse in der Version 3.6.1(Helios) eingesetzt. Verwendet wurde hier die J2EE Variante der Entwicklungsumgebung, da hier bereits alle Funktionen vorhanden sind, die für die Entwicklung von JSP Anwendungen benötigt werden.

4.2.2 Apache Tomcat

Für die Entwicklung wurde der Servlet-Container Apache Tomcat in der Version 6 verwendet und in die Entwicklungsumgebung eingebunden. Tomcat in dieser Version wurde deshalb genutzt, da er auch die spätere Produktionsplattform darstellt und somit zur Vorgabe gehörte.

4.2.3 Oracle JDBC-Treiber

Für den Zugriff auf das Oracle-DBS wurde der Oracle-JDBC-Treiber verwendet, dieser wurde in die Tomcat Umgebung eingebunden.

4.2.4 Oracle SQL-Developer

Um auf dem Oracle-DBS den vollen Funktionsumfang nutzen zu können kam der SQL-Developer zum Einsatz. Dieser wurde von Oracle selbst entwickelt und kann kostenfrei verwendet werden. Zudem ist er plattformunabhängig, da er auf Java Basis entwickelt wurde.

4.3 Konzeption

In diesem Kapitel werden die konzeptionellen Vorüberlegungen dargestellt, die vor der eigentlichen Implementierung getroffen wurden.

4.3.1 Allgemeiner Programmablauf

Der allgemeine Programmablauf des PL/SQL Trainers wird so aussehen, dass der Anwender zunächst auf die Startseite gelangt, auf dieser hat er nun die Möglichkeit eine neue Test-Sitzung zu starten. Nachdem die Sitzung initialisiert wurde, wird er auf die Eingabeseite weitergeleitet, hier wird der Text der zu lösenden Aufgabe dargestellt sowie eine Eingabemöglichkeit für die Lösung angeboten. Der Anwender hat an dieser Stelle nun die Möglichkeiten, entweder seinen Lösungstext einzugeben und, durch Anklicken des „senden“ Buttons, diesen auf Richtigkeit prüfen zu lassen, durch Betätigung des „weiter“ Buttons zur nächsten Frage zu wechseln, und damit die aktuelle Aufgabe zu überspringen, oder die Anwendung, über den „beenden“ Button, zu verlassen und wieder zurück zur Startseite zu gelangen. Bei der Validierung der Benutzereingabe wird der, als Lösung angegebene, Trigger zunächst in die Datenbank geschrieben, sollten hierbei schon Fehlermeldungen seitens der Datenbank zurückgegeben werden, wird die Fehlermeldung auf der aktuellen Seite angezeigt und der Anwender kann seine Eingabe korrigieren. Wird die Eingabe des Benutzers akzeptiert und der Trigger erfolgreich der Datenbank hinzugefügt, folgt als nächster Validierungsschritt der eigentliche Funktionalitätstest. Hierbei werden entsprechende SQL-Statements ausgeführt, die den Trigger zur Auslösung bringen und der Tabelleninhalt, einer für die Aufgabe festgelegten Prüftabelle, wird mit einer Musterlösung verglichen. Nach Abschluss der Validierung wird der Benutzer auf die Ergebnisseite weitergeleitet, hier wird angezeigt ob die angegebene Lösung korrekt war. Hier werden jetzt die Optionen angeboten, auf die Eingabeseite mit der aktuellen Frage zurück zu wechseln, um einen erneuten Lösungsversuch zu tätigen, oder mit der Bearbeitung der nächsten Frage fortzufahren. Nach der Bearbeitung der letzten Aufgabe wird die Anzahl, der vom Benutzer korrekt beantworteten Fragen, in einer Auswertungsseite dargestellt. Der beschriebene Ablauf ist in der folgenden Abbildung nochmals graphisch dargestellt.

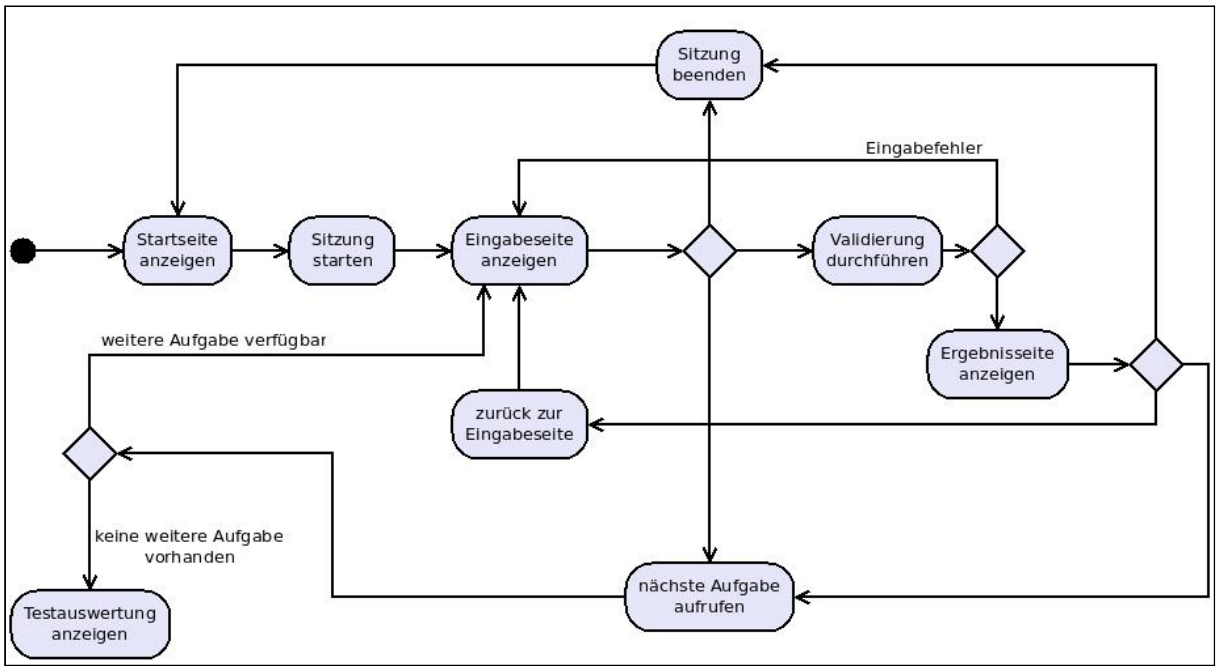


Abbildung 23: Allgemeiner Programmablauf

4.3.2 Anwendungsarchitektur

Für die Entwicklung der Applikation wurde die JSP Model 2 Architekur[7] verwendet. Es sieht eine modularen Aufbau vor, in dem die Geschäftslogik von der Präsentation getrennt wird und durch eine Kontrollinstanz miteinander gekoppelt wird. So wird die Benutzerschnittstelle der Applikation durch die JSPs abgebildet, welche ausschließlich die Darstellungslogik enthalten, während sich die eigentliche Programmlogik in einer separaten Modell-Klasse befindet. Innerhalb der Modell-Klasse finden ebenfalls die Datenbankzugriffe statt, diese werden über die JDBC Schnittstelle realisiert. Die Steuerung der Benutzeraktionen findet innerhalb eines zentralen Controller Servlets statt. Dieses nimmt die Anfragen der JSPs entgegen, führt die Programmlogik aus und leitet die von dem Modellobjekt zurückgegebenen Beans, an die zur Anzeige vorgesehenen JSPs weiter. Die Komponenten der Applikation werden weiterhin in drei Schichten aufgeteilt, die Präsentationschicht, Anwendungsschicht und Persistenzschicht. Die Präsentationsschicht stellt die Benutzerschnittstelle dar und wird durch die JSP Seiten repräsentiert. Zur Anwendungsschicht zählt die Modell-Klasse, die Java-Bean Klassen sowie das Controller Servlet, wobei dieses prinzipiell auch der Präsentationsschicht zugeordnet werden kann, da es die Kopplung der beiden Schichten darstellt und somit übergreifende Funktionen übernimmt. Die Persistenzschicht wird durch das DBS dargestellt, welches das physische Datenmodell beinhaltet.

[7] Vgl. Kapitel 2.4.4

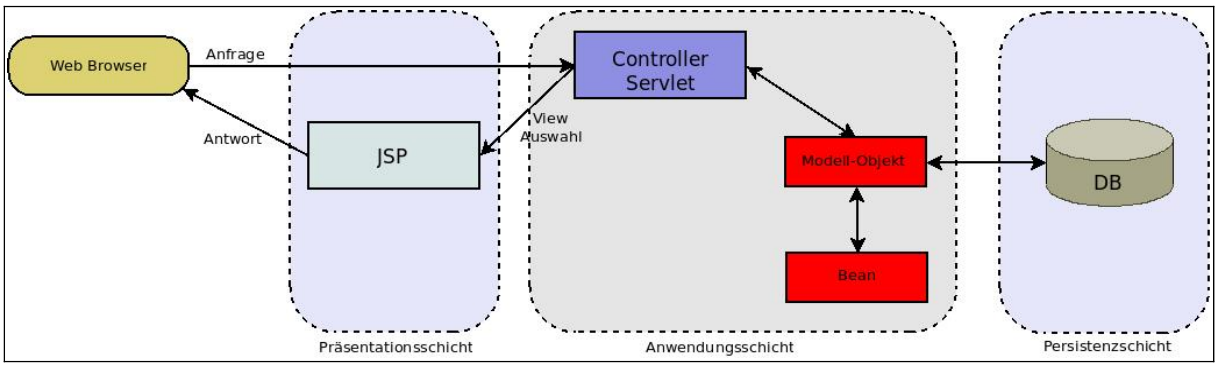


Abbildung 24: Anwendungsarchitektur

4.3.3 Sitzungsverwaltung

Um einen Mehrbenutzerbetrieb innerhalb der Applikation zu unterstützen muss gewährleistet sein, dass jede Sitzung in einem eigenen Kontext läuft und innerhalb dieses Kontextes ein eigenes Datenmodell zur Verfügung gestellt wird.

Eine mögliche Lösung für dieses Problem wäre mehrere Benutzer auf Datenbankebene anzulegen und jeder Sitzung jeweils einen explizit zuzuweisen. Diese Lösung wäre jedoch vor allen Dingen sehr unflexibel und würde weiterhin einen hohen administrativen Aufwand verursachen.

Eine weitere Lösungsmöglichkeit wäre das dynamische Erzeugen von Datenbank Benutzern, hierzu würde die Applikation allerdings DBA Berechtigungen verwenden müssen, was wiederum ein Sicherheitsrisiko darstellt.

Bei der hier gewählten Lösung werden die zum Datenmodell zugehörigen Elemente sozusagen mehrfach nebeneinander, unter Vergabe von, jeweils einer Sitzung zuzuordnenden, dynamischen Bezeichnungen, erstellt. Um eine eindeutige Zuordnung zu gewährleisten, wird für jede Sitzung ein Modellobjekt erzeugt, dem bei der Erstellung eine eindeutige Sitzungs-ID zugewiesen wird. Diese Sitzungs-IDs werden durch ein, in der Anwendung einmalig existierendes, Objekt vergeben. Da jeglicher Zugriff auf das DBS durch das jeweilige Modellobjekt erfolgt, wird in allen auszuführenden SQL-Statements die zugewiesene ID den Datenbank-Objekten als Anhängsel hinzufügt, zum Beispiel aus „select * from Tabellenname“ wird dann „select * from Tabellenname_ID“. Ein Modellobjekt wird dadurch einem bestimmten Satz von Tabellen zugeordnet und alle Datenbankzugriffe innerhalb einer Sitzung

werden auf die für sie vorgesehenen Tabellen umgeleitet, ohne dass der Benutzer davon Kenntnis nimmt. Dieses Verfahren muss selbstverständlich sowohl bei der Erzeugung, des für eine Sitzung gültigen Datenmodells, als auch auf den vom Benutzer als Aufgabenlösung eingegebenen PL/SQL Code angewandt werden. Es stellt somit eine Art Parser dar, wodurch eine saubere Trennung der Sitzungen auf Datenbankebene gewährleistet wird. In der folgenden Abbildung ist dieses Verfahren nochmals veranschaulicht.

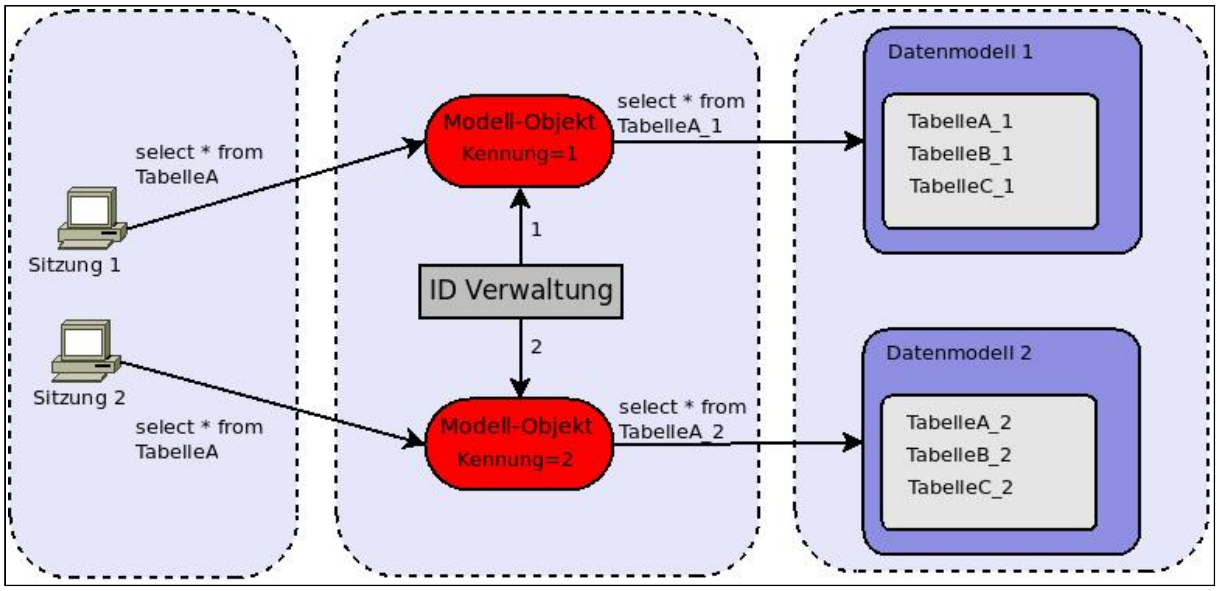


Abbildung 25: Umleiten der SQL-Statements

4.3.4 Dynamischer Auf- und Abbau des Datenmodells

Da für jede Sitzung innerhalb der Anwendung ein eigenes Datenmodell zur Verfügung stehen muss, ist es notwendig, dass bei der Eröffnung einer neuen Sitzung ein entsprechendes Datenmodell erzeugt und im Gegenzug bei Beendigung dieser, das erstellte Modell auch wieder zerstört wird. Dazu werden jeweils für den Aufbau sowie den Abbau des Modells entsprechende SQL-Skripts verwendet. Die Skripts zur Erzeugung des Datenmodells werden ausgeführt, wenn durch den Benutzer die Initialisierung einer neuen Testsitzung gestartet wird. Als problematisch stellt sich allerdings die spätere Zerstörung des Datenmodells dar, wobei hier die eigentliche Durchführung weniger ein Problem darstellt, sondern sich eher in Hinblick auf den richtigen Zeitpunkt der Zerstörung bezieht. Der Anwender hat zwar die Möglichkeit die Sitzung manuell, durch Betätigung des entsprechenden Buttons, zu beenden, muss dies aber nicht zwangsläufig tun. Wenn der Benutzer das Browserfenster mit der Anwendung schließt, erhält die Applikation auf Serverseite keinerlei Rückmeldung darüber.

Dies ist darauf zurückzuführen, dass Webanwendungen das zustandslose HTTP Protokoll verwenden und somit nach dem Request-Response Prinzip arbeiten. Dieses Problem kann jedoch durch dem Umstand gelöst werden, dass jede Sitzung innerhalb eines Webservers, nach Ablauf einer konfigurierbaren Zeitspanne der Inaktivität, ein Timeout Ereignis auslöst. Dieses Session Timeout Ereignis kann wiederum von der Applikation durch die Implementierung eines Listeners abgefangen werden, dieser löst an dieser Stelle den Zerstörungsmechanismus aus.

4.3.5 Verfahren zur Prüfung der Benutzereingaben

Zur Validierung der vom Benutzer eingegebenen Lösungen muss die Eingabe des Anwenders, die einen Datenbank-Trigger darstellt, zunächst der Datenbank hinzugefügt werden. Im Anschluss daran wird der Trigger durch eine oder mehrere DML- Aktionen zur Ausführung gebracht und der Inhalt einer Tabelle, die für die entsprechende Aufgabe zur Prüfung herangezogen werden soll, wird ausgelesen und zwischengespeichert. Die Angabe mehrerer DML-Statements ist hier optional anzusehen, ist jedoch für Aufgabenstellungen sinnvoll, in denen der Trigger nur unter speziellen Bedingungen entsprechende Aktionen durchführen soll.

Eine Möglichkeit den nun selektierten Tabelleninhalt zu Prüfen wäre es, zu jeder Aufgabe den Inhalt der entsprechenden Prüftabelle als Musterlösung zu speichern und zu Vergleichszwecken heranzuziehen. Bei dieser Lösung wäre jedoch der Aufwand, der beim Einpflegen neuer Aufgaben betrieben werden müsste, extrem hoch und würde die Verwendung zusätzlicher Tools unumgänglich machen.

Bei der hier gewählten Lösung wird für jede Aufgabe ein entsprechender Trigger definiert, der die Musterlösung darstellt. Dieser Trigger wird ebenfalls in die Datenbank geschrieben, zur Auslösung gebracht und der Inhalt der Prüftabelle wird nun ein zweites mal ausgelesen und zwischengespeichert. Die beiden gespeicherten Tabelleninhalte werden nun miteinander verglichen. Sind die Inhalte identisch gilt die Aufgabe als richtig gelöst. Die nachstehende Abbildung veranschaulicht diese Verfahrensweise.

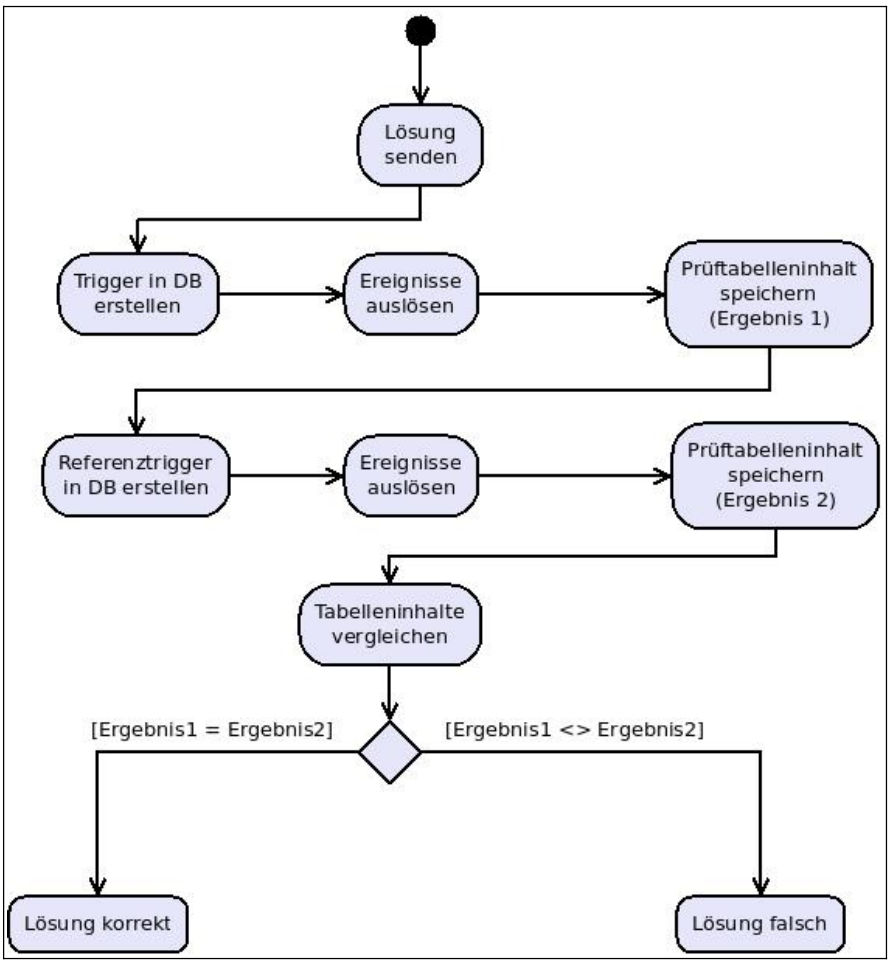


Abbildung 26: Validierung der Benutzereingaben

Für Aufgaben, die durch einen Trigger gelöst werden sollen, der bei gewissen Ereignissen eine Ausnahme auslöst und die Verarbeitung abbricht, muss das Verfahren zur Lösungsvalidierung entsprechend angepasst werden. Hierzu werden für die Aufgabe zwei auslösende Statements definiert, wobei das erste Statement den Trigger zwar anspricht, jedoch nicht die in der Aufgabe spezifizierte Ausnahme auslöst, dies geschieht erst durch Ausführung des zweiten Statements. Bei dieser Art von Aufgaben ist die Angabe zweier Statements also zwingend erforderlich. Nur so ist es möglich zu prüfen, ob der Benutzer in seiner Lösung die Ausnahmebedingungen richtig programmiert hat. Wurde die Ausnahme bereits durch das erste Statement ausgelöst, gilt die Lösung als falsch, ebenso wenn die Ausnahme durch keines der beiden Statements ausgelöst wurde. Wurde die Ausnahme korrekt ausgelöst, werden die Ausnahmemeldungen aus beiden Durchläufen miteinander verglichen. Sind sie identisch wurde die Aufgabe richtig gelöst. Diese angepasste Verfahrensweise ist in der folgenden Abbildung dargestellt.

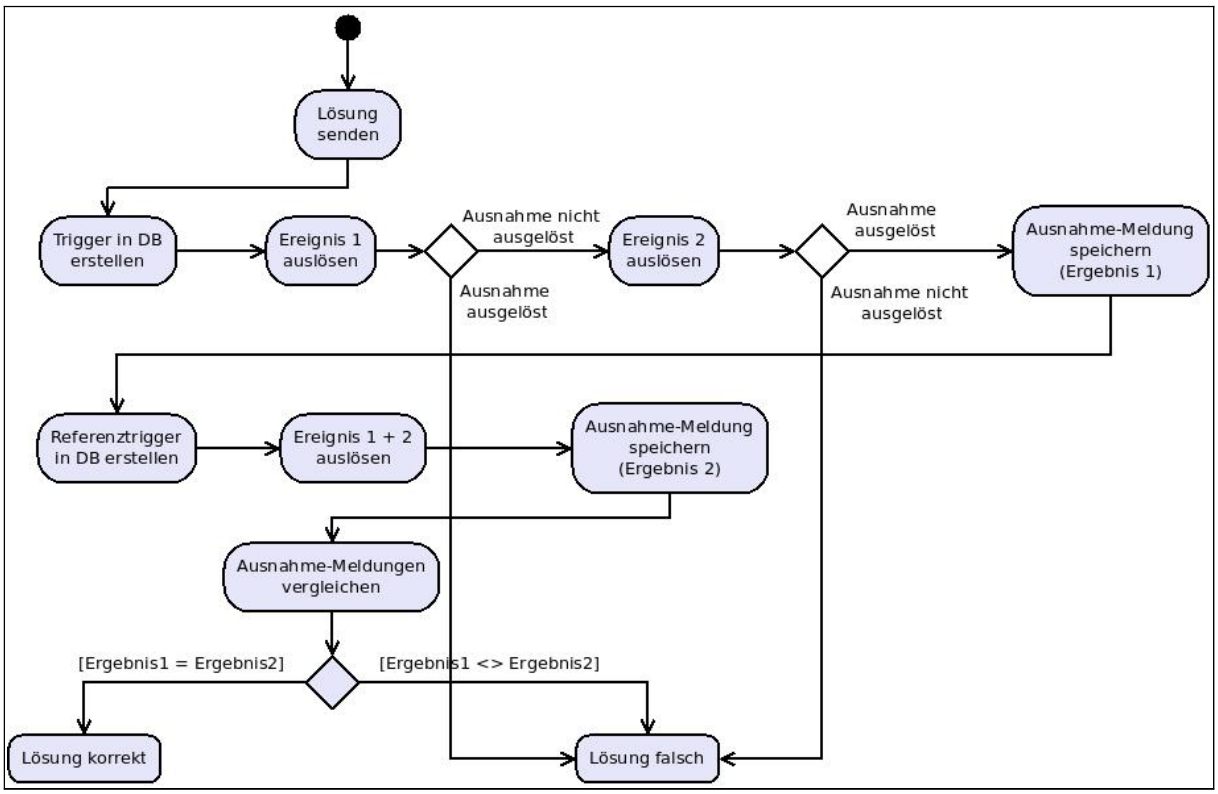


Abbildung 27: Validierung Ausnahmebehandlung

4.3.6 Vermeidung von SQL-Injections

Da die Eingaben des Anwenders ausführbaren SQL-Code enthalten, muss der Code vor Ausführung auf der Datenbank zunächst entsprechend geprüft werden, so dass sichergestellt ist, dass keine nicht vorgesehene Aktionen durchgeführt werden. Aus diesem Grunde wird der eingegebene Text zunächst darauf geprüft, ob er einer syntaktisch gültigen Definition eines PL/SQL Datenbanktriggers entspricht. Weiterhin dürfen innerhalb der vom Anwender definierten SQL Statements nur die Tabellen angesprochen werden, die in dem für den Benutzer erzeugten Datenmodell vorhanden sind. Auf Tabellen wie beispielsweise die, die zur die Speicherung der Aufgaben verwendet werden, muss jeglicher Zugriff strikt unterbunden werden. Um dies zu gewährleisten enthält die Applikation eine Liste von Bezeichnern, die im Lösungstext des Benutzers nicht enthalten sein dürfen. Der Text wird nach jedem dieser Einträge durchsucht und erst dann wenn diese Prüfung negativ ausfällt, darf der PL/SQL Code an das DBS gesendet werden.

4.3.7 Sicherstellung der Datenkonsistenz

Innerhalb des Validierungsprozesses werden an verschiedensten Stellen Tabelleninhalte des

Datenmodells verändert, zum einen durch die DML-Statements, die ausgeführt werden um den Trigger zur Auslösung zu bringen, zum anderen durch den Trigger selbst. Dies findet im Validierungsprozess an zwei Punkten statt, da hier einmal der Trigger des Anwenders ausgeführt wird und im Anschluss noch der Trigger der Musterlösung. Hierbei ist von essentieller Bedeutung, dass vor jeder Auslösung der Datenbestand aller zum Modell gehörigen Tabellen dem ursprünglichen Zustand entspricht.

Eine mögliche Lösung für dieses Problem wäre es, nach jedem Auslösungsvorgang und Speicherung des Prüftabelleninhalts alle Tabelleninhalte wieder herzustellen. Dies hätte jedoch den Nachteil, dass auch die Tabellen wiederhergestellt werden, in denen keinerlei Änderung stattgefunden hat und somit unnötig Systemressourcen verbraucht werden.

In der für die Anwendung gewählte Lösung wird jeweils der SQL-Code der feuernden Ereignisse sowie der PL/SQL Code des jeweiligen Triggers nach allen Tabellenbezeichnungen durchsucht, die in dem verwendeten Datenmodell enthalten sind. Die gefundenen Tabellennamen werden in eine Liste aufgenommen, da die Inhalte dieser Tabellen, nach Auslösung des jeweiligen Triggers, potentiell verändert sein könnten. Die in der Liste enthaltenen Tabellen werden, nach der Triggerausführung, inhaltlich wiederhergestellt. Die folgende Abbildung stellt dieses Verfahren dar.

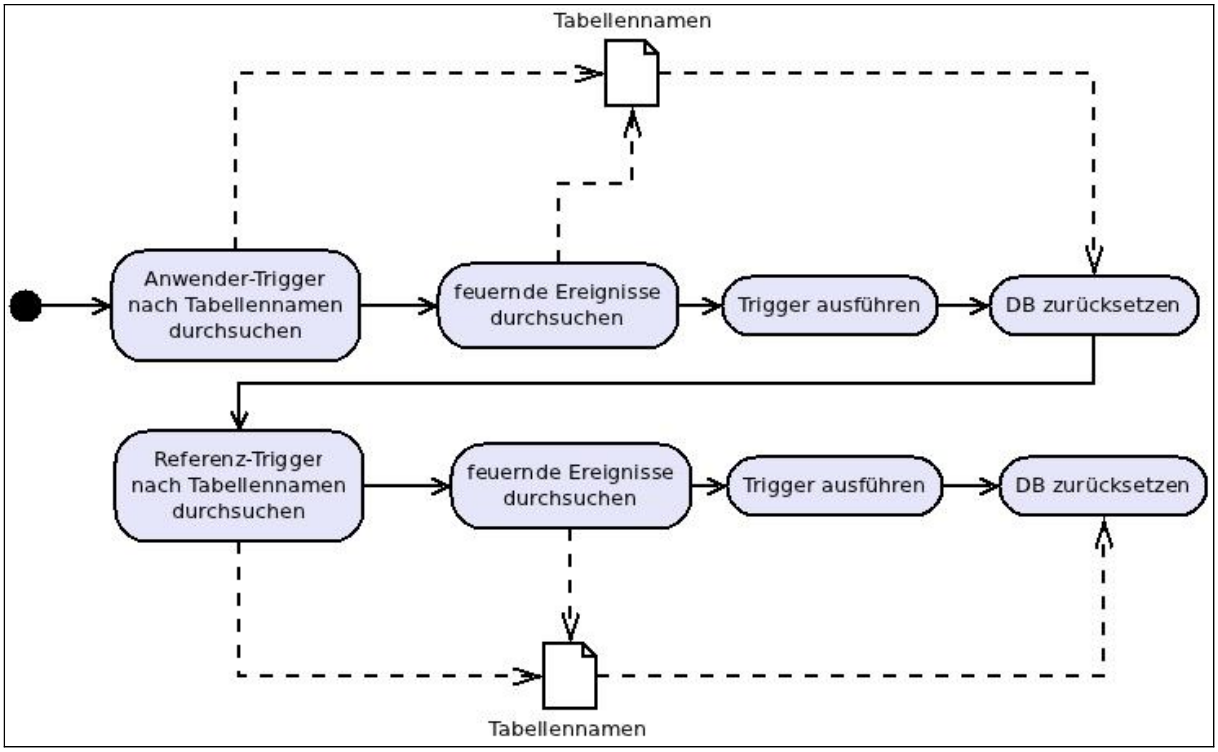


Abbildung 28: Sicherstellung der Datenkonsistenz

4.3.8 Übungsaufgaben

Um dem Benutzer die Möglichkeit zu geben seine Kenntnisse im Bereich der Triggerprogrammierung zu testen, mussten für die Trainingsanwendung geeignete Aufgaben definiert werden. Im Rahmen des Projekts wurden zu diesem Zweck Aufgaben entwickelt, die alle klassischen Aufgabengebiete von Triggern abdecken. Die Aufgaben untergliedern sich in die folgenden Anwendungsfälle:

- Konsistenzprüfung mit Fehlerkorrektur
- Konsistenzprüfung mit Fehlerausgabe
- Pflege anhängiger Tabellen
- Protokollierung und Archivierung

Als Grundlage für das Datenmodell wurde das FahrradDB Modell gewählt, das auch in den Vorlesungen im Fach DuI verwendet wird und den Benutzern bereits bekannt ist. Dieses relationale Datenbank-Modell beschreibt die Relationen eines Fahrradhändlers zu dessen Kunden, Lieferanten, Angestellten sowie der Lagerverwaltung. Es beinhaltet genügend Möglichkeiten um für nahezu alle genannten Aufgabengebiete entsprechende Aufgaben zu definieren, einzig für das Aufgabengebiet der „Protokollierung und Archivierung“ musste das Datenmodell noch um entsprechende Protokolltabellen erweitert werden.

4.4 Implementierung

4.4.1 Persistenzschicht

Die Persistenzschicht der Applikation besteht aus den Tabellen des Datenmodells, dass für den Testablauf verwendet wird. Die Tabellen dieses Modells werden für jede Sitzung dynamisch erstellt und bei Beendigung wieder zerstört. Ein weiterer Teil der Persistenzschicht stellen die applikationseigenen Tabellen dar.

4.4.1.1 Testdatenmodell

Wie in Kapitel 4.3.8 bereits erwähnt wird für die Testdurchführung das FahrradDB Modell als Grundlage verwendet und um die beiden Tabellen „Positionsarchiv“ und „Gehaltsprotokoll“ erweitert. Das vollständige relationale Modell ist in der folgenden Abbildung dargestellt.

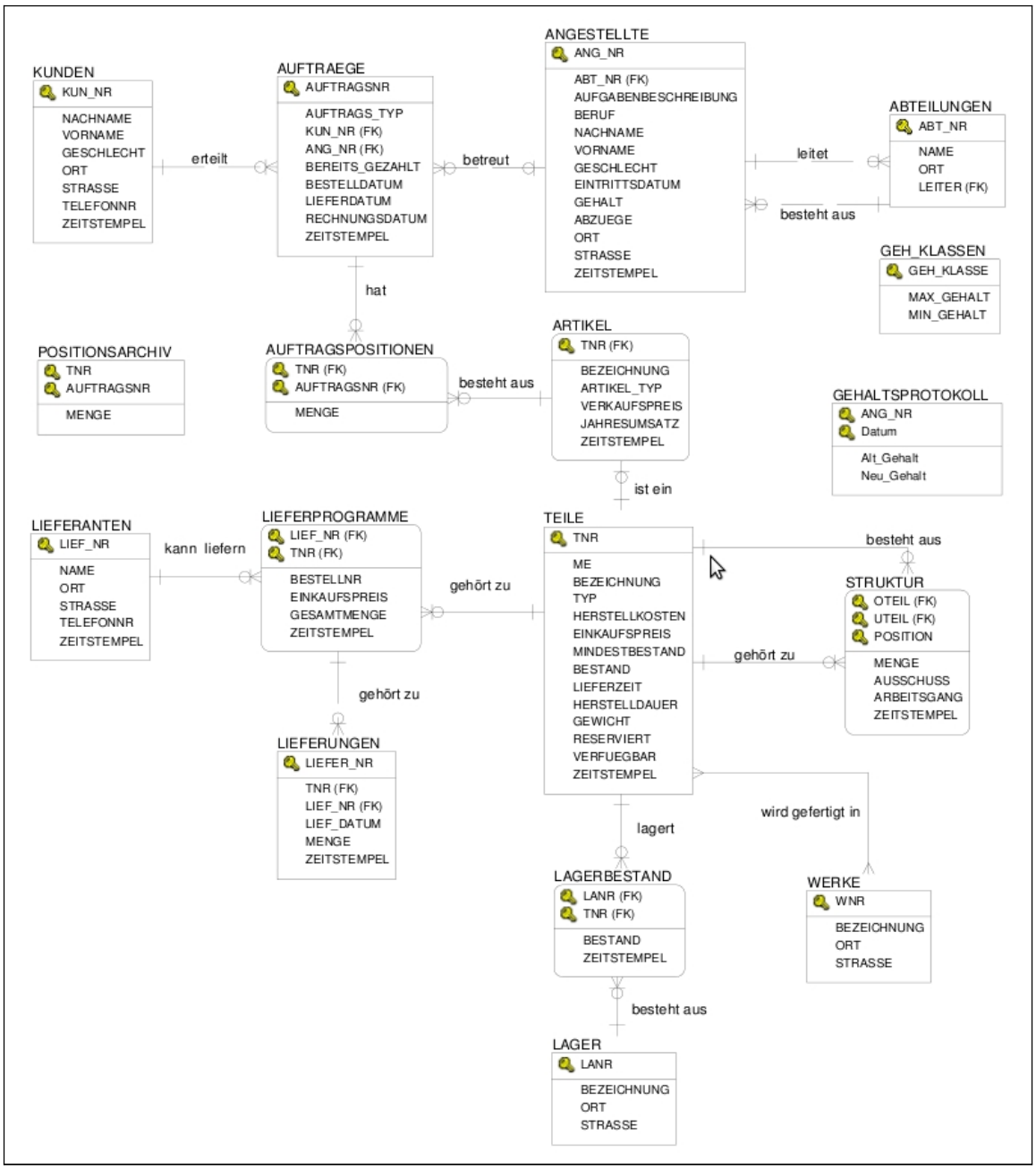


Abbildung 29: Test-Datenmodell

In diesem Datenmodell existieren nun konkrete Möglichkeiten zur Erstellung von Übungsaufgaben für die verschiedenen Anwendungsbereiche von Datenbanktriggern. Im Folgenden werden nun einige dieser Möglichkeiten, die für die Erstellung der Übungsaufgaben verwendet wurden, beschrieben.

Konsistenzprüfung mit Fehlerkorrektur

Abteilungsleiter sollen einer bestimmten Mindestgehaltsgruppe angehören. Wenn sich der Leiter einer Abteilung ändert oder ein neuer Datensatz in der Tabelle Abteilungen erstellt wird, muss die Gehaltsgruppe des Leiters geprüft werden und gegebenenfalls auf das

Mindestgehalt dieser Gehaltsgruppe in der Tabelle Angestellte erhöht werden.

Die Summe aller Gehälter darf ein bestimmtes Budget nicht überschreiten. Bei Einstellung eines neuen Mitarbeiters, also hinzufügen eines Datensatzes in der Tabelle Angestellte, soll geprüft werden ob das vorgegebene Budget überschritten wurde, ist dies der Fall so sollen alle Gehälter, die eine bestimmte Höhe überschreiten, auf einen festgelegten, niedrigeren Wert korrigiert werden.

Konsistenzprüfung mit Fehlerausgabe

Gehälter dürfen nicht sinken. Wird in der Tabelle Angestellte in einer Zeile das Feld Gehalt geändert, muss der alte mit dem neuen Wert verglichen werden. Ist der neue Wert geringer soll die weitere Verarbeitung abgebrochen werden und eine Ausnahme ausgelöst werden.

Pflege abhängiger Tabellen

Die Tabellen Teile und Lagerbestand besitzen jeweils das Feld Bestand. Diese beiden Felder stehen in Abhängigkeit zueinander, die Tabelle Lagerbestand enthält den Bestand eines Teiles für jedes einzelne Lager in dem es verfügbar ist, die Tabelle Teile enthält die Summe dieser Bestände. Somit muss bei einer Bestandsänderung in der Tabelle Lagerbestand auch die Tabelle Teile aktualisiert werden.

Protokollierung und Archivierung

Wenn ein Datensatz in der Tabelle Auftragspositionen gelöscht wird, sollen die gelöschten Datensätze in der Tabelle Positionsarchiv archiviert werden.

Wenn in der Angestelltentabelle Änderungen in der Spalte "Gehalt" vorgenommen werden, soll die Veränderung, also vorheriges Gehalt und neues Gehalt sowie das aktuelle Tagesdatum, in der Tabelle "Gehaltsprotokoll" protokolliert werden.

4.4.1.2 Applikationseigene Tabellen

Zusätzlich zu den Tabellen des Test-Datenmodells werden in der Applikation noch weitere Tabellen zur internen Verwendung benötigt.

Tabelle Questions **Tabelle FiringStatements**

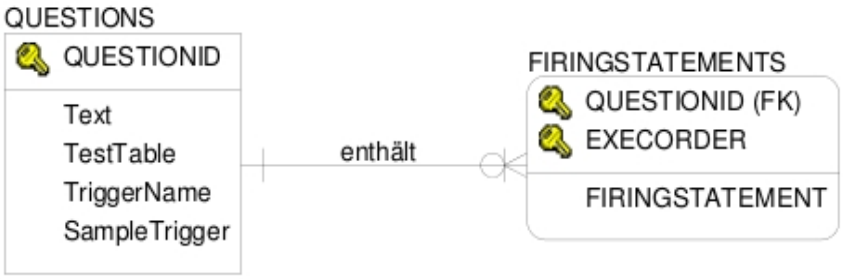


Abbildung 30: Tabelle Questions und FiringStatements

Die Tabelle *Questions* enthält die Informationen zu den Test-Aufgaben. Jede Aufgabe besitzt eine eindeutige *QUESTIONID*, die den Primärschlüssel der Tabelle darstellt. Weiterhin beinhaltet sie den Aufgabentext, der dem Benutzer angezeigt wird. Im Attribut *TestTable* wird ein Tabellenname des Test-Datenmodells spezifiziert, anhand der die Lösungsprüfung durchgeführt wird. In *TriggerName* wird angegeben unter welchem Namen der vom Benutzer erstellte Lösungstrigger erzeugt werden soll und *SampleTrigger* enthält die Definition eines Triggers, der die Musterlösung der Aufgabe darstellt. Weiterhin besteht zu der Tabelle *FiringStatements* eine 1 zu n Beziehung, diese beinhaltet zu jeder Aufgabe der *Questions* Tabelle, im Attribut FiringStatement, ein oder mehrere SQL-Statements, die bei der Lösungsvalidierung zur Auslösung der Trigger verwendet werden. Das Attribut *ExecOrder* gibt hierbei die Reihenfolge der Ausführung an.

Tabelle TableNames

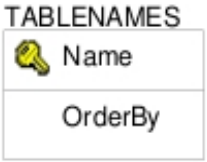


Abbildung 31: Tabelle: TableNames

Die Tabelle *TableNames* enthält alle Tabellennamen des Test-Datenmodells und zu jeder dieser Tabellen wird im Attribut *OrderBy* das Sortierungskriterium der jeweiligen Tabelle angegeben.

Tabelle DBItems

Abbildung 32: Tabelle: DBItems

In der Tabelle *DBItems* werden alle restlichen Datenbankobjekte, die bei der Erstellung des Test-Datenmodells mit erzeugt werden, angegeben. Darunter fallen alle Constraints, Indexes, Views und Sequences des Modells.

4.4.2 Anwendungsschicht

4.4.2.1 „Controller“ Servlet

Das Controller Servlet stellt die Kopplung zwischen der Präsentationsschicht und der Modellobjekte dar. Alle Anfragen der JSPs werden, mit entsprechenden Parametern, an dieses Servlet übergeben und in der „doGet“ Methode ausgewertet und weiterverarbeitet. Die Methode verarbeitet die folgenden Anfragen.

- Neue Sitzung starten
- Validierung der Benutzereingabe
- Sitzung beenden
- nächste Aufgabe
- Tabelleninhalt anzeigen
- Tabellenliste anzeigen
- vorherige Aufgabe

Der prinzipielle Ablauf der Anfrageverarbeitung ist bei allen Anfragen gleich. Es wird immer das Modellobjekt, aus dem Session-Scope der anfragenden Sitzung, abgerufen und je nach Art der Anfrage die entsprechenden Methoden des Objekt ausgeführt. Die einzige Ausnahme stellt hier die Aktion zum Start einer neuen Sitzung dar, da hier das Modellobjekt zunächst noch instantiiert werden muss. Danach werden die von dem Modellobjekt zurückgegebenen Java-Beans im Request-Scope abgelegt und die Anfrage wird auf die zur Anzeige vorgesehene JSP Seite weitergeleitet. Hier werden die Attribute der Beans ausgelesen, eventuell noch

aufbereitet und dargestellt.

4.4.2.2 Klasse SessionIDManager

Durch die Klasse *SessionIDManager* werden Funktionalitäten implementiert um für jede Sitzung ein eindeutige ID bereitzustellen. Dieses Objekt beinhaltet ein Integer Array, dessen Indexpositionen die Sitzungs-IDs darstellen. Die Größe des Arrays wird festgelegt durch den im Konstruktor übergebenen Parameter. Die Werte der Indexpositionen nehmen hier die Werte „0“ oder „1“, zur Kennzeichnung ob die ID verfügbar ist oder zur Zeit verwendet wird, an. Die Methode *getFreeSessionID()* gibt immer die erste Indexposition zurück die mit dem Wert „0“ belegt ist. Sind bereits alle IDs in Verwendung, liefert die Methode den Wert „-1“ zurück. Durch Aufruf der Methode *releaseSessionID(int)* wird die hier übergebene ID wieder freigegeben.

4.4.2.3 „Modell“ Klasse DBManager

Die Klasse *DBManager* beinhaltet die eigentliche Anwendungslogik und stellt somit die komplexeste Klasse der Applikation dar. Sie stellt die Verbindung zur Datenbank her, beinhaltet Funktionen zur Erstellung und Zerstörung des Datenmodells und führt die Validierung der Benutzereingaben durch Im Folgenden werden hier die wichtigsten Methoden, mit Bezug zu den in Kapitel 4.3 vorgestellten Konzepten, dieser Klasse beschrieben.

Konstruktur – DBManager(String scriptPath, String dbConnection, int sessionID)

Dem Konstruktor der Klasse werden zwei Parameter übergeben, zum einen der Pfad, in dem sich die SQL-Skripts zur Erstellung und Zerstörung des Test-Datenmodells befinden, ein Verbindungsparameter zur Herstellung der JDBC- Datenbankverbindung und eine Sitzungs-ID. Innerhalb des Konstruktors wird die Datenbankverbindung initialisiert, das Datenmodell erzeugt und die Daten der ersten Aufgabe geladen.

Umleiten der SQL-Statements - replaceDBItemNames(String sql, List<String> itemList)

In dieser Methode wird der übergebene String nach den Einträgen in der Liste durchsucht und den gefundenen Einträgen innerhalb des Strings wird die, im Konstruktor übergebene, Sitzungskennung als Anhängsel hinzugefügt. Durch diese Methode wird das in Kapitel 4.3.3 beschriebene Konzept zur Kopplung eines Modellobjekts an ein bestimmtes Datenmodell implementiert.

Ausführen eines SQL-Skripts - executeSQLScript(String filePath)

Die Methode *executeSQLScript* führt das im Übergabeparameter angegebene Skript aus. Dazu wird jede SQL Anweisung einzeln gelesen und zunächst durch die Methode *replaceDBItemNames* für die Sitzung entsprechend vorbereitet. Die Liste, die an die Methode *replaceDBItemNames* übergeben wird, enthält zum einen die Einträge aus der Tabelle *TableNames* und weiterhin die aus der *DBItems* Tabelle, dies hat den Hintergrund da innerhalb der auszuführenden SQL-Skripte zur Erstellung des Datenmodells neben den Modell-Tabellen auch alle weiteren Objekte wie beispielsweise Constraints und Indexes erzeugt werden, diese müssen ebenfalls eindeutig zugeordnet werden. Die aufbereiteten Statements werden dann über die JDBC Schnittstelle an das DBS gesendet.

Datenmodell erzeugen - CreateDatabase()

Hier werden die SQL Skripte für die Erstellung des Datenmodells, unter Verwendung der Methode *executeSQLScript*, ausgeführt.

Datenmodell zerstören - destroyDatabase()

Diese Methode führt die SQL-Skripte zur Zerstörung des, für die Sitzung erstellte, Datenmodells aus wofür ebenfalls die *executeSQLScript* Methode eingesetzt wird.

Ermitteln von potentiell veränderten Tabellen - checkForModifiedTables(String sql)

Der übergebene String wird nach enthaltenen Tabellennamen des Test-Datenmodells durchsucht, wobei alle vorkommenden in eine Liste aufgenommen werden.

Tabelleninhalte wiederherstellen - RestoreTableContent()

Mit dieser Methode werden für alle Tabellen, die innerhalb der Methode *checkForModifiedTables* in die Liste aufgenommen wurden, die ursprünglichen Inhalte wiederhergestellt. Im Zusammenspiel mit der Methode *checkForModifiedTables* wird hier das in Kapitel 4.3.7 beschriebene Konzept zur Sicherstellung der Datenkonsistenz implementiert.

Syntaktische Prüfung - checkTriggerDefinition(String trigger)

Hier wird geprüft ob es sich bei dem als Parameter übergebene Trigger um eine syntaktisch korrekte Definition eines PL/SQL Triggers handelt und ob der Trigger unter dem in der Aufgabenstellung angegebenen Namen definiert wurde.

Prüfung auf ungültige Bezeichner - checkForInvalidItemNames(String trigger)

Die Funktion prüft, ob der übergebene Trigger Tabellennamen enthält, auf die der Benutzer keinen direkten Zugriff haben soll. Zusammen mit der Methode *checkTriggerDefinition* stellt sie die Umsetzung des Konzepts zur Vermeidung von SQL-Injections dar, welches in Kapitel 4.3.6 beschrieben wurde.

Tabelleninhalt abfragen - getTableData(String tabName)

Durch Aufruf dieser Methode wird der Inhalt der als Parameter übergebene Tabellenname in Form einer Array List vom Typ String zurückgegeben. Der selektierte Tabelleninhalt wird hier nach dem, in der Tabelle *TableNames* im *OrderBy* Attribut, definierten Kriterium sortiert.

Trigger ausführen - executeTrigger(String trigger)

Die *executeTrigger* Methode wird dazu verwendet einen übergebenen Trigger zur Ausführung zu bringen und den Tabelleninhalt der Prüftabelle zurückzugeben.

Validierung der Benutzereingabe - validateTrigger(String trigger)

Die Methode *validateTrigger* übernimmt den vom Benutzer eingegebenen Lösungstrigger als Eingabeparameter. Hier findet nun der Validierungsprozess, konzeptionell dargestellt in Kapitel 4.3.5, statt. Dies ist die Methode, die durch das *ControllerServlet* zur Lösungsvalidierung verwendet wird und stellt sozusagen die Kernfunktionalität der Klasse dar.

Die Funktionsweisen der Methoden zur Validierung der Benutzereingaben, zur Ausführung der Trigger sowie der Initialisierung eines Klassenobjekts werden in Kapitel 4.4.2.6 noch genauer betrachtet.

4.4.2.4 Java Beans

Innerhalb der Applikation wurden diverse Java-Bean-Klassen implementiert, die dienen zur Speicherung von Daten und besteht ausschließlich aus Membervariablen mit entsprechenden get und set Methoden.

Question

Das *QuestionBean* bildet einen Datensatz der Tabelle *Questions* ab. Sie dient also dazu die Informationen zu einer Aufgabe zu speichern und enthält so wie die bereits beschriebene

Tabelle die Attribute *ID, Text, TestTable, TriggerName* und *SampleTrigger*. Weiterhin enthält sie das Attribut *FiringStatements,* welches als Listenobjekt definiert ist. Es enthält die mit der jeweiligen Aufgabe verknüpften Inhalte des *FiringStatement* Attributs aus der Tabelle *FiringStatements*.

Result

Das *ResultBean* wird dazu verwendet, das Ergebnis eines Validierungsvorgangs zu speichern. Es besteht aus den Array Lists *queryResponse*, *expectedResponse* und dem Integerwert *success*. *QueryResponse* wird dazu verwendet den Inhalt der Prüftabelle, der durch Ausführung des vom Anwender als Lösung eingegebenen Triggers entsteht, zu speichern, während *expectedResponse* den Inhalt der Tabelle darstellt, der durch den Trigger der Musterlösung entsteht. Die Variable *success* wird mit dem Wert „1“ belegt, wenn die beiden ArrayLists den gleichen Inhalt aufweisen und mit dem Wert „0“, wenn dies nicht der Fall ist.

ModifiedFlag

Diese Bean Klasse enthält die beiden Variablen *tableName* und *isModified*. In *tableName* wird ein Tabellenname gespeichert und *isModified* stellt einen Indikator dar, der angibt ob die Tabelle einer eventuellen inhaltlichen Änderung unterliegen könnte. Die Klasse *DBManager* enthält ein Listenobjekt, in der für jede Tabelle des Test-Datenmodells ein Objekt dieser Klasse existiert.

4.4.2.5 Hilfsfunktionen

Zusätzlich zu den bereits beschriebenen Klassen, wurden noch weitere Hilfsmethoden implementiert, diese wurden als statische Methoden in der Klasse *JSPHelper* angelegt. Diese Funktionen werden nun erläutert.

transformToHTMLTableRows(List <String> stringList)

Diese Methode dient dazu eine Liste vom Typ String, wobei jeder String Wert mehrere Werte durch Semikon getrennt beinhaltet, in eine HTML Tabelle zu transformieren.

compareStringArrays(List<String> list1, List<String> list2)

In Java ist es möglich einzelne String Objekte auf inhaltliche Gleichheit zu prüfen, was jedoch nicht für Listen von String Objekten gilt. Da diese Funktionalität jedoch benötigt wird um die Rückgabewerte der *executeTrigger* Methode der Klasse *DBManager* miteinander zu

vergleichen, wurde diese Hilfsfunktion entwickelt.
Um die Array Lists auf inhaltliche Gleichheit zu prüfen, wird zunächst geprüft ob die beiden Arrays die gleiche Anzahl von Objekten aufweisen, ist dies der Fall werden die Inhalte der Indexpositionen der Arrays miteinander verglichen. Die Methode gibt im Falle der inhaltlichen Gleichheit *TRUE* zurück, andernfalls *FALSE*.

convertToLowerCaseSQL(String code)
In den SQL-Statements, die der Benutzer innerhalb seiner Lösung definiert, müssen die verwendeten Tabellenbezeichnungen in die für diese Sitzung vorgesehenen Tabellennamen umgewandelt werden. Dazu wird der Text nach allen im Datenmodell vorhandenen Tabellennamen durchsucht. Problematisch hierbei ist jedoch, dass SQL wie auch PL/SQL Code nicht Case-Sensitiv behandelt wird, String Vergleiche in Java aber schon. Um dieses Problem zu lösen musste der Lösungstext des Benutzer zunächst speziell aufbereitet werden. Zur besseren Verarbeitung werden zunächst alle Tabulator Zeichen durch Leerzeichen ersetzt und im Anschluss alle doppelten Leerzeichen entfernt. Die Zeichen innerhalb des Textes werden nun auf Kleinschreibung geändert, das heißt alle bis auf die, die in einfache Hochkommata eingeschlossen sind, da sie, innerhalb von SQL sowie PL/SQL, keinen Code sondern Text repräsentieren.

4.4.2.6 Interaktion der Komponenten
Hier wird nun das Zusammenspiel der beschriebenen Applikationsobjekte dargestellt. Zum besseren Verständnis wird dies anhand von Anwendungsfällen beschrieben.

Initialisierung einer Test Sitzung
Zur Initialisierung einer neuen Test-Sitzung wird zunächst, durch Aufruf der Methode *getfreeSessionID()* des *SessionIDManagers* Objekts, eine Sitzungs-ID abgerufen. Die ID wird nun verwendet um ein *DBManager* Objekt zu instantiieren, indem sie bei der Erzeugung des Objekts als Parameter übergeben wird. Für jede Sitzung wird ein solches Objekt erzeugt und im jeweiligen *Session-Scope* abgelegt damit es im gesamten Lebenszyklus der Sitzung verfügbar bleibt. Im Konstruktor des *DBManager* Objekts wird nun, durch die Methode *connect()*, eine Verbindung zur Datenbank hergestellt und im Anschluss, durch *createDatabase()*, das Datenmodell für diese Sitzung erzeugt. Der Aufbau des Datenmodells stellt den zeitaufwändigsten Part der Initialisierung dar. Auf dem Entwicklungssystem nahm die Erzeugung des Modells durchschnittlich drei bis vier Sekunden in Anspruch. Im letzten Schritt der Konstruktion, werden die Daten der ersten Aufgabe aus der Datenbank abgerufen

und in einem QuestionBean Objekt gespeichert. Das QuestionBean wird innerhalb der Methode *nextQuestion()*, über die jeweiligen *set* Methoden, mit Daten befüllt. Hiermit ist die Initialisierung des *DBManager* Objekts abgeschlossen. Das Controller Servlet ruft nun die Methode *getQuestion()* des *DBManager* Objekts auf und bekommt dadurch das *QuestionBean* als Rückgabewert geliefert. Das Bean wird nun zur Präsentation an eine JSP weitergegeben.

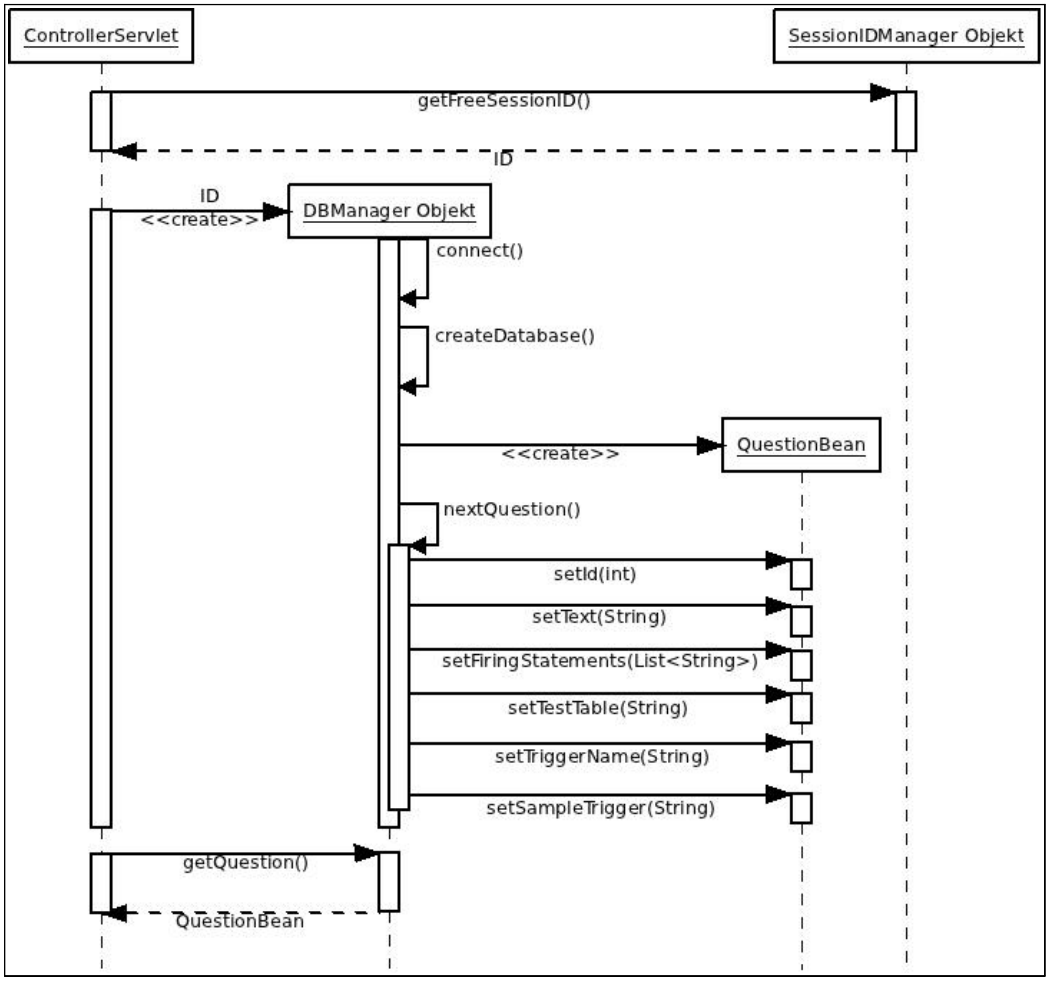


Abbildung 33: Sequenzdiagramm: Initialisierung einer Test-Sitzung

Validierung einer Benutzeingabe

Bei der Validierung einer Benutzereingabe wird der vom Benutzer eingegebene Text an das *ControllerServlet* übergeben. Das Servlet ruft nun die *validateTrigger* Methode des für die betreffende Sitzung erzeugten *DBManager* Objekts auf, die den Eingabetext als Parameter übernimmt. Das *DBManager* Objekt verwendet nun die statische Methode *convertToLowerCaseSQL* der *JSPHelper* Klasse zur Formatierung des PL/SQL Codes. Als Nächstes folgt nun die Prüfung des Codes durch die Methoden *checkTriggerDefinition* und

checkForInvalidItemNames. Waren diese Prüfungen erfolgreich wird nun, zur Speicherung der inhaltlichen Validierungsergebnisse, ein *ResultBean* Objekt instantiiert. Der Trigger wird jetzt durch die *executeTrigger* Methode zur Ausführung gebracht und liefert den Inhalt der Prüftabelle zurück. Die Verfahrensweise der Triggerausführung wird im Anschluss noch detaillierter betrachtet. Der zurückgegebene Tabelleninhalt wird durch die Methode *setQueryRespone* an das *ResultBean* übergeben und dort gespeichert. Das gleiche Verfahren wird nun ebenfalls mit dem, in der aktuellen Aufgabe definierten, Trigger durchgeführt. Dazu wird durch den Methodenaufruf *getSampleTrigger* des *QuestionBean* Objekts der Code des Triggers abgerufen und als Übergabeparameter an die *executeTrigger* Methode übergeben. Der Rückgabewert wird über die *setExpectedRespone* Methode in das *ResultBean* übertragen. Jetzt können die beiden Tabelleninhalte durch Zuhilfenahme der Methode *compareStringArrays* der *JSPHelper* Klasse miteinander verglichen werden, das Ergebnis hieraus wird wiederum im *ResultBean* gespeichert. Damit ist die Validierung abgeschlossen und das *ResultBean* Objekt wird an das *ControllerServlet* zurückgegeben.

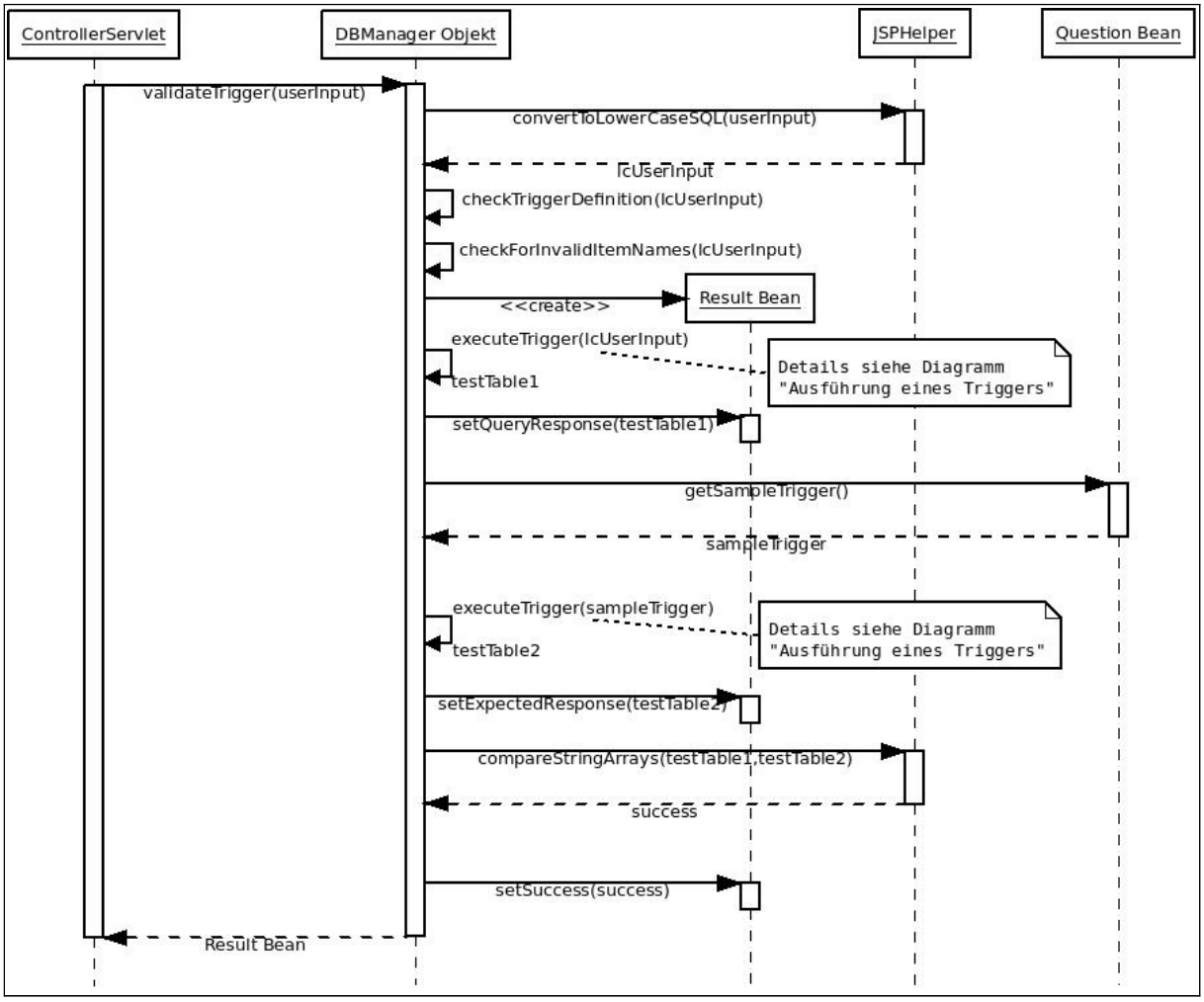


Abbildung 34: Sequenzdiagramm: Validierung einer Benutzereingabe

Ausführung eines Triggers

An dieser Stelle soll die Systematik zur Ausführung der Trigger, die in der *executeTrigger* Methode stattfindet, nochmals im Detail betrachtet werden.

Der *executeTrigger* Methode wird der zur Ausführung vorgesehene Triggercode als Parameter übergeben. Hier werden zunächst durch die Methode *checkForModifiedTables*, mit Übergabe des Triggercodes, alle Tabellenbezeichnungen in eine Liste aufgenommen, die im Trigger enthalten sind. Das gleiche wird mit den auslösenden SQL-Statements gemacht, die zuvor jedoch noch aus der *QuestionBean*, durch die Methode *getFiringStatements*, abgerufen werden müssen. Jetzt kommt die Methode *replaceDBItemNames* sowohl für den Trigger selbst, als auch für die auslösenden Statements zum Einsatz. Der Trigger wird nun an die Datenbank gesendet und im Anschluss die auslösenden SQL-Statement abgesetzt. Aus der *QuestionBean* wird nun der Name der Tabelle abgefragt, die zur Ergebnisprüfung herangezogen werden soll, dies geschieht über die Methode *getTestTableName*. An dieser Stelle wird über die Methode *getTableData*, die den zuvor ermittelten Tabellennamen als Parameter übernimmt, der Inhalt der Prüftabelle abgerufen. Bevor der Tabelleninhalt an den Aufrufer zurückgegeben wird, werden die Tabellen, die zuvor in die Liste der zu wiederherstellenden Tabellen aufgenommen wurden, durch die Methode *restoreTableContent* wieder in ihren ursprünglichen Zustand zurückversetzt.

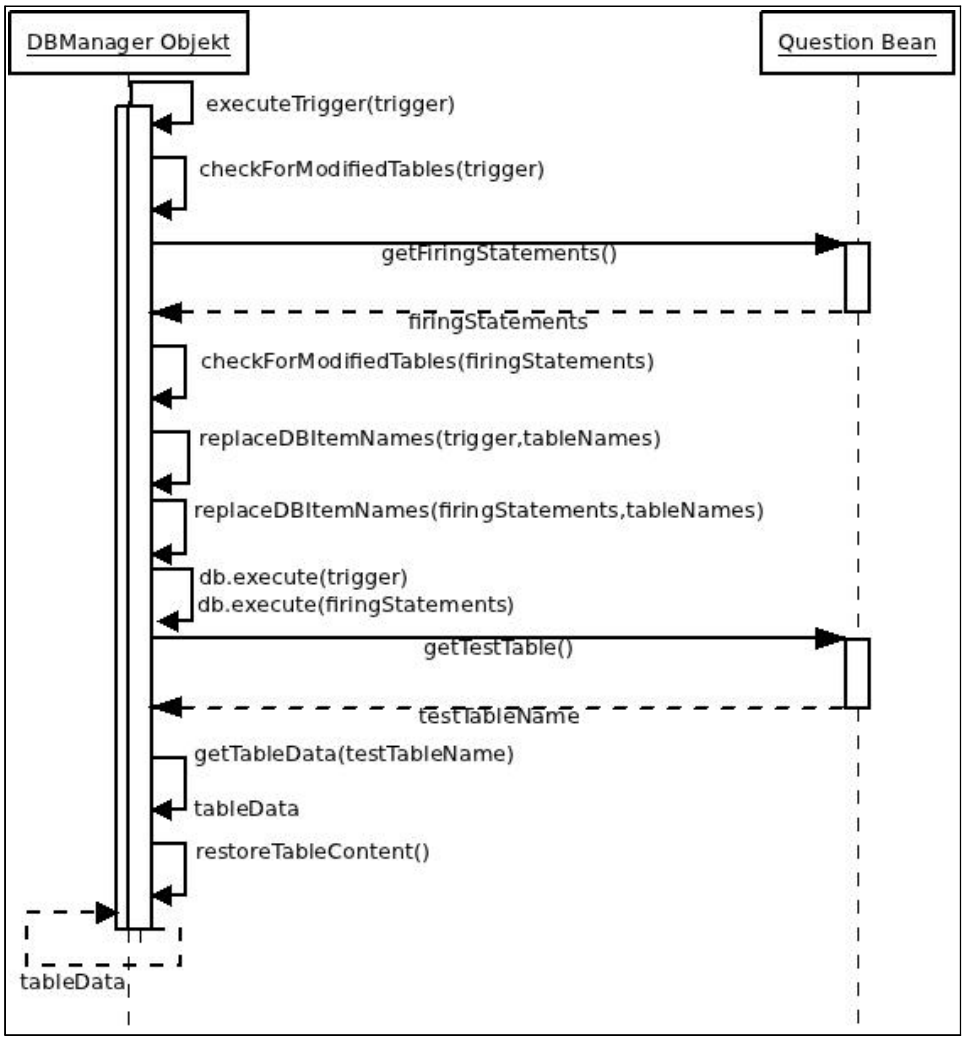


Abbildung 35: Sequenzdiagramm: Ausführung eines Triggers

4.4.2.7 Listener - DBSessionListener

Sowohl bei manueller Beendigung der Sitzung durch den Anwender als auch nach Ablauf der Sitzung nach einer gewissen Zeitspanne der Inaktivität, muss das entsprechende Datenmodell der betreffenden Sitzung zerstört werden. Dies wurde in Kapitel 4.3.4 bereits angesprochen. Zur Implementierung dieser Funktionalität wurde die Klasse DBSessionListener erstellt, um auf das Ereignis eines Session-Timeouts entsprechend zu reagieren. Die hierfür erstellte Klasse wurde von der Basisklasse HttpSessionListener abgeleitet, in dieser kann nun durch überschreiben der Methode sessionDestroyed die Implementierung vorgenommen. Damit der Listener auch von der Applikation verwendet wird ist jedoch noch ein weiterer Schnitt notwendig, er muss innerhalb der Anwendung noch registriert werden. Dazu wurde im Deployment Descriptor folgender Eintrag hinzugefügt.

```
<listener>
  <listener-class>plsqltrainer.DBSessionListener</listener-class>
</listener>
```

Abbildung 36: Registrierung: DBSessionListener

Somit ist der Listener in der Applikation bekannt und wird bei Anwendungsstart aktiviert.

4.4.2.8 Spezielle Implementierungsaspekte

Nebenläufigkeitsproblem bei Zuweisung der Sitzungskennungen

Wie bereits in Kapitel 4.4.2.2 beschrieben wird durch die Klasse *SessionIDManager* die Vergabe von eindeutigen Sitzungs-IDs implementiert. Durch Aufruf der Methode *getFreeSessionID()* wird eine solche ID zurückgeliefert. Problematisch hierbei ist die Thread Sicherheit, da mehrere Sitzungen gleichzeitig auf dieses Objekt zugreifen können und Servlets standardmäßig nicht Thread sicher sind. So kann es unter bestimmten Umständen vorkommen, dass zwei Sitzungen die gleiche ID zugewiesen wird. Die nachfolgende Abbildung stellt die kritische Code Stelle dar.

```
public int getFreeSessionID () {
        for (int i=0;i<ids.length;i++) {
                if (ids[i] == 0) {
                        ids[i] = 1;
                        return i;
                }
        }
        return -1;
}
```

Kritischer Ausführungspunkt

Abbildung 37: Nebenläufigkeitsproblem: kritischer Ausführungspunkt

Wird an der gekennzeichneten Stelle die Verarbeitung des einen Threads unterbrochen und an einen Anderen abgegeben, der die gleiche Funktion ausführt kommt zu dem geschilderten Problem. Zur Vermeidung des Problems muss sichergestellt sein, dass die ein Thread während der Ausführung dieser Funktion nicht unterbrochen wird, dies wurde folgendermaßen implementiert.

```
public synchronized int getFreeSessionID () {
        for (int i=0;i<ids.length;i++) {
                if (ids[i] == 0) {
                        ids[i] = 1;
                        return i;
```

Abbildung 38: Nebenläufigkeitsproblem: Lösung

Herstellen der Datenbankverbindung

Die Datenbankanbindung an die Oracle Datenbank wird über die JDBC Schnittstelle realisiert, innerhalb des Java Quellcodes könnte die Erstellung einer solchen Verbindung so aussehen.

```
try {
    Class.forName( "oracle.jdbc.OracleDriver" );
    Connection con =  DriverManager.getConnection("jdbc:oracle:thin:@host:1521:xe","user","pw");

 } catch ( ClassNotFoundException e ) {
    System.err.println( "Keine Treiber-Klasse!" );
 }
```

Abbildung 39: Erstellung JDBC Verbindung

Es ist allerdings nicht sonderlich vorteilhaft, die Verbindungsinformationen zur Datenbank fest im Quellcode zu verdrahten, da jedes mal Anpassungen im Code nötig währen, falls beispielsweise das Passwort eines Benutzers geändert wird. Daher wird hier die Möglichkeit verwendet, die Verbindungsinformationen über die JNDI- Schnittstelle zur Verfügung zu stellen. Dazu wurde im Verzeichnis META-INF die XML Datei context.xml angelegt, sie enthält nun die Verbindungsinformationen. Die XML Datei weist folgenden Inhalt auf.

```
<Context>
   <Resource
        name="jdbc/plsqltrainer"
        auth="Container"
        type="javax.sql.DataSource"
        factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory"
        maxActive="30" maxIdle="10" maxWait="10000"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@localhost:1521:xe"
        username="username"
        password="password"
        connectionProperties=""
        description="Oracle Datasource"/>
</Context>
```

Abbildung 40: JNDI Konfiguration: context.xml

Jetzt muss der hier definierte Name noch im Deployment Descriptor der Applikation bekannt gemacht werden, dies geschieht durch das Hinzufügen des folgenden Eintrags.

```
 <resource-ref>
  <description>DB Connection</description>
  <res-ref-name>jdbc/plsqltrainer</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Abbildung 41: JNDI Konfiguration: Deployment Descriptor

Innerhalb des Java Quellcodes wird die Verbindung zur Datenbank nun folgendermaßen initialisiert.

```
Context initCtx = new InitialContext();
db = (DataSource) initCtx.lookup("java:comp/env/jdbc/plsqltrainer");
v_con = db.getConnection();
```

Abbildung 42: JNDI: Verbingsaufbau

So ist es möglich, die Verbindungsparameter in der XML Datei anzupassen, ohne Änderungen im Quellcode vornehmen zu müssen.

Problematik bei Wiederherstellung der Tabelleninhalte

In Kapitel 4.3.7 wird das konzeptionelle Verfahren zur Sicherstellung der Integrität der Tabellen beschrieben. Implementiert wird dieses Verfahren in der Klasse *DBManager* durch die Methoden *checkForModifiedTables* und *RestoreTableContent*. Als problematisch hat sich hierbei jedoch, bei der Implementierung der *RestoreTableContent* Methode, der Umstand erwiesen, dass die Tabelleninhalte der neu zu füllenden Tabellen zunächst gelöscht werden müssen. Dies stellt in so fern ein Problem dar, dass Datensätze nicht einfach gelöscht werden dürfen, wenn zu ihnen in anderen Tabellen Fremdschlüsselbeziehungen existieren. Entweder wird die Löschaktion vom DBS verweigert oder im Fall eines „on delete cascade“ Constraints wird der Datensatz in der Tabelle zu der die Beziehung besteht ebenfalls gelöscht, was wiederum einen inkonsistenten Datenbestand zu Folge hätte. Dieses Problem wurde dadurch gelöst vor der Durchführung der Inhaltswiederherstellung alle foreign-key Constraints des betreffenden Datenmodells deaktiviert und am Anschluss wieder aktiviert werden. Dazu wurden entsprechende SQL-Skripte erstellt, Auszüge daraus zeigen die folgenden Abbildungen.

Auszug des Skripts zum deaktivieren der foreign-key Constraints

```
ALTER TABLE Teile_Werke DISABLE CONSTRAINT T_WE_T_FK;
ALTER TABLE Teile_Werke DISABLE CONSTRAINT T_WE_WE_FK;
ALTER TABLE Struktur DISABLE CONSTRAINT S_T_FK2;
ALTER TABLE Struktur DISABLE CONSTRAINT S_T_FK;
```

Abbildung 43: Auszug SQL-Skript: DisableConstraints

Auszug des Skript zur Reaktivierung der Constraints

```
ALTER TABLE Angestellte ENABLE CONSTRAINT ANG_ABT_FK;
ALTER TABLE Artikel ENABLE CONSTRAINT AR_T_FK;
ALTER TABLE Auftraege ENABLE CONSTRAINT AU_ANG_FK;
ALTER TABLE Auftraege ENABLE CONSTRAINT AU_KUN_FK;
```

Abbildung 44: Auszug SQL-Skript: EnableConstraints

Zur Ausführung der Skripts wurde die Klasse *DBManager* um die Methode *disableConstraints* und analog dazu um die Methode *enableConstraints* erweitert. Diese wurden dann in der Methode *RestoreTableContent* verwendet.

Bereinigen von Artefakten im Fehlerfall

Bei der Ausführung von JSP Anwendungen kann es unter bestimmten Konstellationen dazu kommen, dass Sitzungen nicht korrekt terminieren. Dies kann der Fall sein beim manuellen Stoppen des Anwendungsservers ober bei Absturz des gesamten Systems. In diesen Fällen greift der unter 4.4.2.4 beschriebene Mechanismus nicht und somit würden die zu diesem Zeitpunkt vorhandenen Tabellen bestehen bleiben. Um dennoch nach dem Neustart der Applikation einen konsistenten Zustand zu gewährleisten, musste ein entsprechender Bereinigungsmechanismus entwickelt werden. Zur Lösung dieses Problems wurde ein weiterer *Listener* implementiert, dieser wurde von der Basisklasse *ServletContextListener* abgeleitet. Durch Überschreiben der Methode *contextInitialised* kann auf das Ereignis des Applikationsstarts reagiert werden. Auch diese *Listener* Klasse wird wiederum im Deployment Descriptor durch folgenden Eintrag registriert.

```
<listener>
   <listener-class>plsqltrainer.DBContextListener</listener-class>
</listener>
```

Abbildung 45: Registrierung: DBContextListener

Die Bereinigungsfunktionalität an sich ist innerhalb einer PL/SQL Prozedur implementiert, die über über die JDBC Schnittstelle aufgerufen wird. Die Prozedur nutzt in ihrer Implementierung die Gegebenheit, dass in Oracle Datenbanken für alle unter einem Datenbank-User erstellte Datenbankobjekte entsprechende Tabellen existieren, aus denen diese selektiert werden können. Die Tabelle „USER_TABLES“ beispielsweise enthält alle Tabellen die für den jeweiligen Benutzer verfügbar sind, ähnliche Tabellen existieren ebenfalls für Constraints, Views sowie Sequences. Aus diesen Tabellen werden nun die Bezeichnungen aller Datenbankobjekte selektiert, die für die Test- Sitzungen erzeugt wurden. Der Löschvorgang der verschiedenen Objekte wird in der folgenden Reihenfolge durchgeführt.

1. Löschung der Constraints
2. Löschung der Tabellen

3. Löschung der Sequences
4. Löschung der Views

Optimierung der Performance

Zur Minimierung der Systemlast, bei der Erzeugung der Datenmodelle, wurde der Erzeugungsmechanismus entsprechend optimiert, so dass nur die Tabellen angelegt werden, die in den Übungsaufgaben auch benutzt werden. Hierzu wurde eine Konfigurationsdatei erstellt, in der zunächst alle Tabellenbezeichnungen enthalten sind. Hier wurden nun die Tabellen auskommentiert, die nicht benötigt werden. Bei der Ausführung der SQL-Skripts zur Erzeugung des Datenmodells, wird nun für jedes SQL-Statement anhand dieser Liste geprüft, ob es an das DBS gesendet werden soll.

Ein weiterer Punkt zur Optimierung der Performance, ist die Begrenzung der maximalen Sitzungsanzahl. Die maximale Anzahl gleichzeitiger Sitzungen, wird durch den im Konstruktor übergebenen Parameter der Klasse *SessionIDManager* gesteuert. Um diesen Parameter dynamisch konfigurieren zu können, wurde ebenfalls eine entsprechende Konfigurationsdatei angelegt.

4.4.3 Präsentationsschicht

4.4.3.1 Startseite - *index.jsp*

Abbildung 46: Startseite: index.jsp

Die *index.jsp* stellt die Startseite der Webanwendung dar, sie enthält lediglich einen Informationstext über die Anwendung sowie einen Button, durch den die eigentliche Trainingsanwendung gestartet wird. Nach Betätigung des Buttons wird eine Aktivitätsanzeige in Form eines Ladebalkens eingeblendet, um dem Anwender eine Rückmeldung darüber zu geben, dass seine Anfrage entgegengenommen wurde. Dies soll vor allem mehrmaliges Betätigen des Buttons, durch den Benutzer, verhindern.

Zur Implementierung des Ladebalkens wurde der JSP zunächst folgender Bereich hinzugefügt.

```
<div id="progressbar" style="display:none">
        Test wird gestartet<br />
        <img src="images/progress.gif" alt="" height="15px" width="250px"/><br />
        Bitte warten...
</div>
```

Abbildung 47: Ladebalken: HTML Bereich

Der Parameter *style="display:none"* gibt an, dass alles was sich innerhalb des „div" Tags befindet nicht angezeigt werden soll. Um den Bereich nach Klicken des Buttons sichtbar zu machen wurde die folgende JavaScript Funktion implementiert.

```
function showProgressBar() {
    document.getElementById("progressbar").style.display = "block";
    return true;
}
```

Abbildung 48: Ladebalken: Java Script Funktion

Hierdurch wird der „style" Parameter des Bereichs von „display:none" auf „display:block" geändert und der definierte Bereich wird eingeblendet.

4.4.3.2 Eingabeseite - *input.jsp*

Abbildung 49: Eingabeseite - input.jsp

Auf dieser Seite wird der Aufgabentext und ein Eingabefeld angezeigt, in dem der Anwender seine Lösung eintragen kann. Hier stehen nun drei Buttons zur Verfügung, der Button „senden“ zum Absenden des Formulars, „weiter“ um zur nächsten Aufgabe zu wechseln oder „beenden“ um die Anwendung abzubrechen und damit zur Startseite zurückzukehren. Weiterhin wird auf dieser Seite, innerhalb eines eines eingebunden Frames, die *schemetables.jsp* angezeigt.

4.4.3.3 Tabellennamen Listenansicht - *schemetables.jsp*

Abbildung 50: Tabellennamen Listenansicht - schemetables.jsp

In der *schemetables.jsp* werden die Bezeichnungen aller im Modell befindlichen Tabellen untereinander dargestellt. Die Tabellennamen werden in Form von HTML- Links angezeigt, die bei Aufruf die *tableview.jsp* in einem neuen Fenster öffnet.

4.4.3.4 Tabellenansicht - *tableview.jsp*

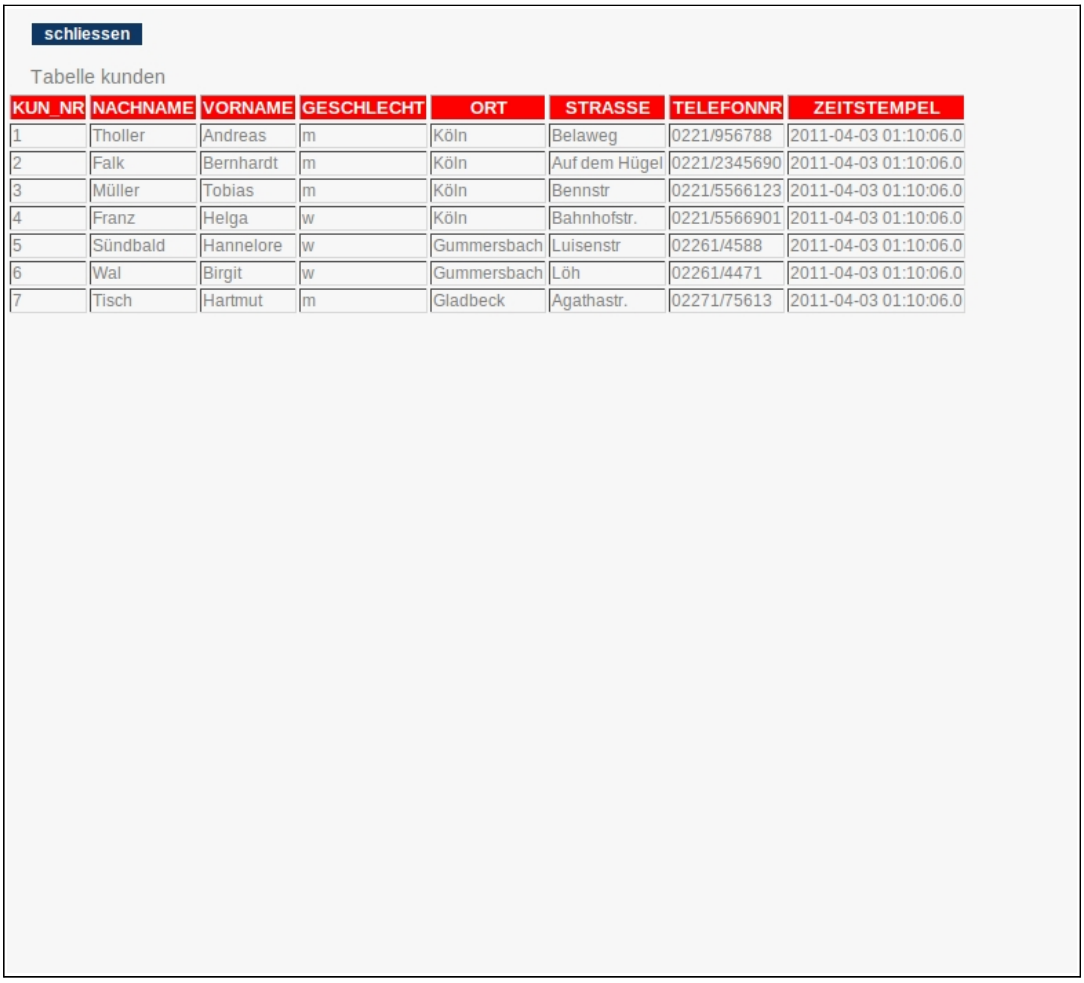
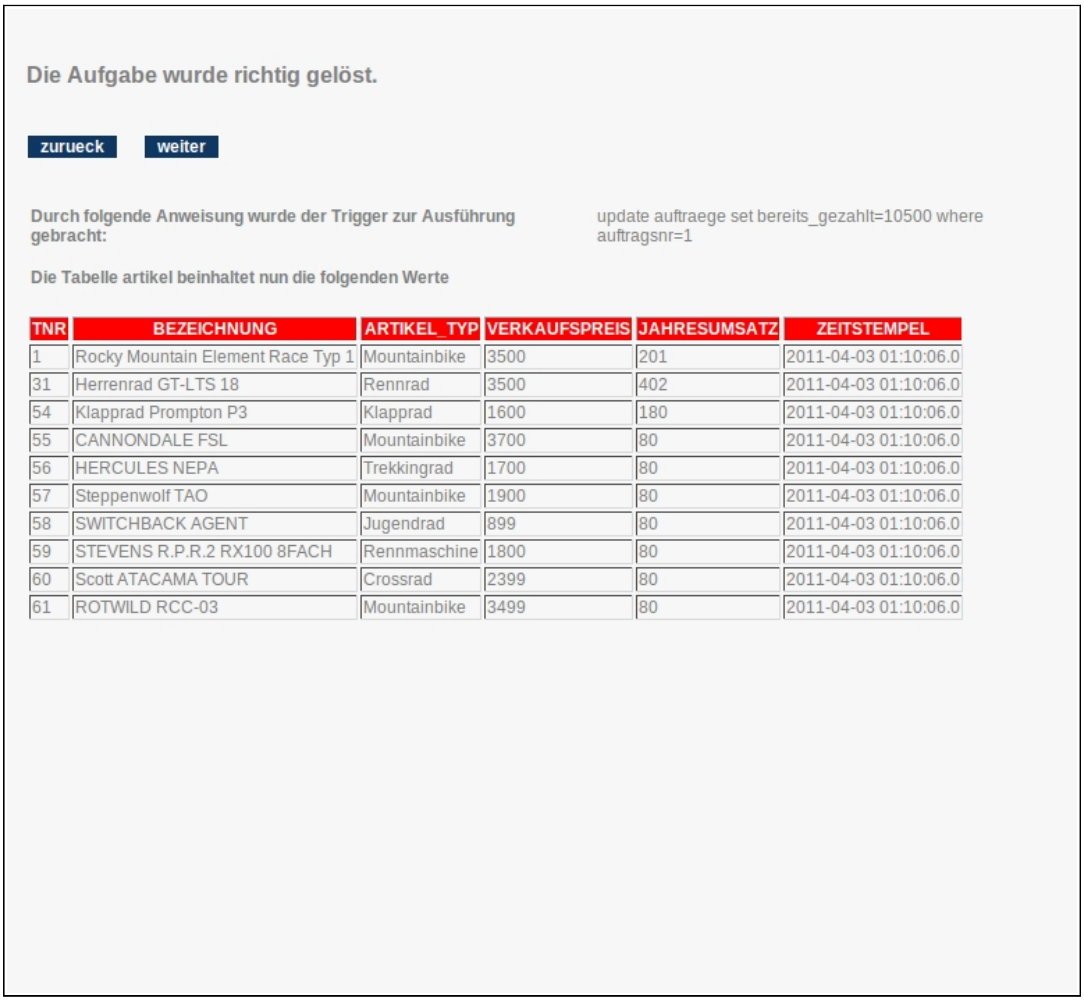


Abbildung 51: Tabellenansicht - tableview.jsp

Die *tableview.jsp* wird dazu verwendet, die Inhalte einer Tabelle darzustellen. Sie erhält die Inhalte in Form eines String Arrays wobei jede Arrayposition eine Tabellenzeile enthält, die Werte sind hierbei jeweils durch Semikolon getrennt. Die Inhalte des Arrays werden ‚durch Verwendung der statischen Methode *transformToHTMLTableRows* der Klasse *JSPHelper*, in eine HTML Tabelle umgewandelt.

4.4.3.5 Ergebnisseite - *result.jsp*



TNR	BEZEICHNUNG	ARTIKEL_TYP	VERKAUFSPREIS	JAHRESUMSATZ	ZEITSTEMPEL
1	Rocky Mountain Element Race Typ 1	Mountainbike	3500	201	2011-04-03 01:10:06.0
31	Herrenrad GT-LTS 18	Rennrad	3500	402	2011-04-03 01:10:06.0
54	Klapprad Prompton P3	Klapprad	1600	180	2011-04-03 01:10:06.0
55	CANNONDALE FSL	Mountainbike	3700	80	2011-04-03 01:10:06.0
56	HERCULES NEPA	Trekkingrad	1700	80	2011-04-03 01:10:06.0
57	Steppenwolf TAO	Mountainbike	1900	80	2011-04-03 01:10:06.0
58	SWITCHBACK AGENT	Jugendrad	899	80	2011-04-03 01:10:06.0
59	STEVENS R.P.R.2 RX100 8FACH	Rennmaschine	1800	80	2011-04-03 01:10:06.0
60	Scott ATACAMA TOUR	Crossrad	2399	80	2011-04-03 01:10:06.0
61	ROTWILD RCC-03	Mountainbike	3499	80	2011-04-03 01:10:06.0

Abbildung 52: Ergebnisseite - result.jsp

Hier wird dem Benutzer das Resultat seiner eingegebenen Lösung präsentiert. Angezeigt wird eine Meldung bezüglich der Korrektheit, die zur Auslösung des Triggers verwendeten SQL-Statements sowie der Inhalt der Ergebnistabelle.

4.4.3.6 Testauswertung – finish.jsp

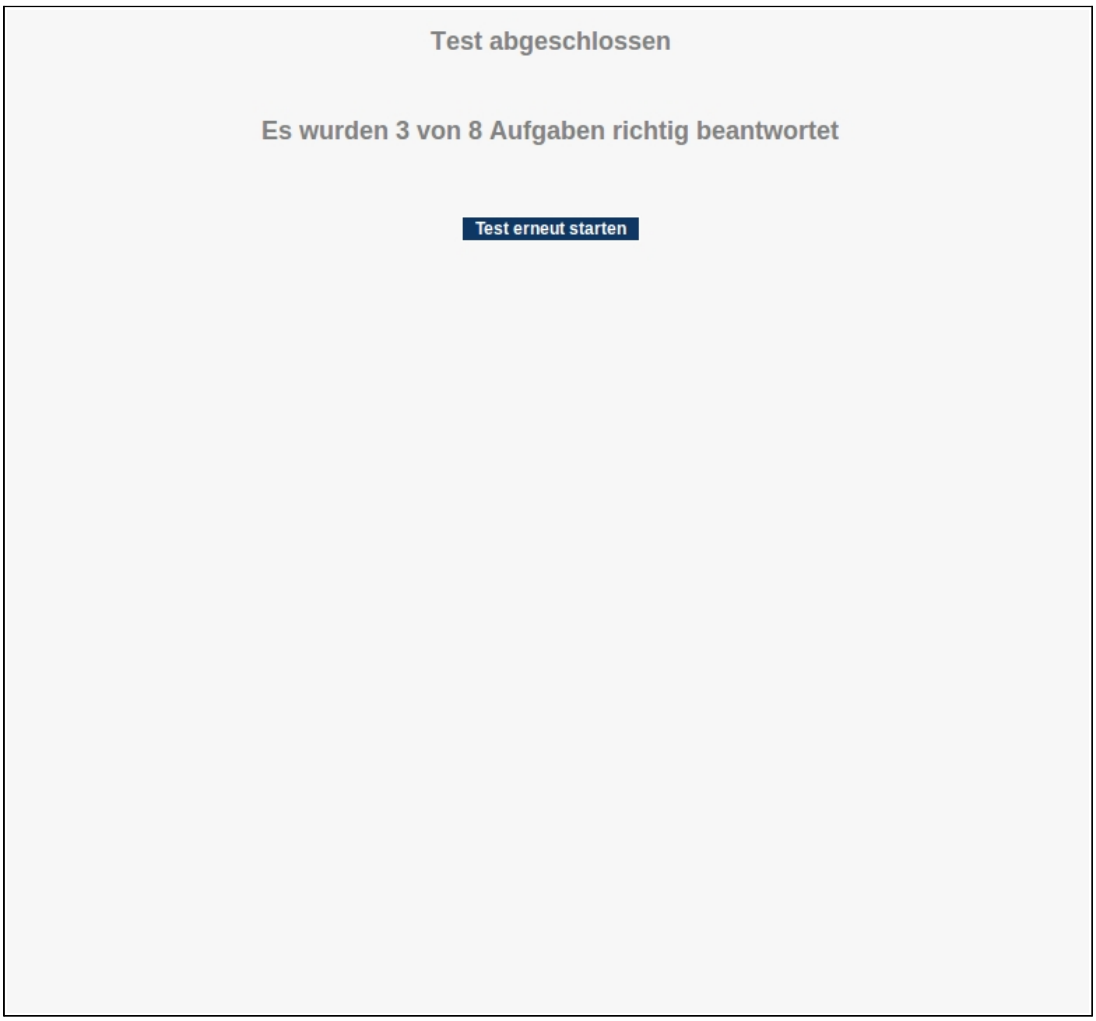


Abbildung 53: Testauswertung – finish.jsp

Nachdem alle Testfragen durchlaufen wurden, wird dem Benutzer zum Testabschluss eine Auswertungsseite angezeigt. Hier wird die Anzahl der richtig gelösten Aufgaben dargestellt und die Möglichkeit angeboten, eine weitere Testsitzung zu starten.

5 Fazit

An dieser Stelle sollen die Ergebnisse, die innerhalb dieser Diplomarbeit entstanden sind, zusammengefasst betrachtet und bewertet werden.

Im ersten Teil der Arbeit wurden, durch Verwendung des eLML Frameworks, zwei Lektionen zum Thema PL/SQL aus dem MS-Power-Point Format in das eLML Format übertragen und daraus dann eine Ausgabe in das HTML Format generiert. Diese Lektionen konnten dadurch in das edb eingebunden werden und stehen somit nun online zur Verfügung. Durch den praktischen Einsatz des eLML Frameworks wurde deutlich, welche Vorteile dessen Einsatz gegenüber herkömmlichen Methoden bietet. Der größte Vorteil besteht darin, dass eLML eine gut durchdachte und sich bereits in der Praxis bewährte allgemeine Struktur vorgibt. Alle mit eLML erstellten Lektionen besitzen einen identischen Grundaufbau und entsprechen somit einem gewissen Standard. Weitere positive Aspekte von eLML sind zum einen die Trennung des Inhalts von der Darstellung, so dass für alle erstellten Lektionen ein einheitliches Layout definiert werden kann und zum anderen die Tatsache, dass es sich um eine Open-Source Lösung handelt und damit keinerlei zusätzliche Kosten entstehen.

Im zweiten Teil wurde eine Trainingsanwendung für den Themenbereich PL/SQL entwickelt, sie wurde als JSP Webanwendung implementiert und ebenfalls in edb eingebunden. Die Hauptproblematik bei der Entwicklung bestand hierbei weniger in der Implementierung der Webanwendung an sich, sondern vielmehr darin geeignete Lösungen zur Prüfung der vom Benutzer als zur Lösung der Aufgaben eingegebenen PL/SQL Code zu finden. Dazu wurden im Verlauf der Arbeit verschiedene konzeptionelle Ansätze gegenübergestellt und weiterhin die Implementierung der gewählten Lösung beschrieben. Die gestellten Anforderungen an die Applikation wurden innerhalb der Diplomarbeit umgesetzt und damit eine Möglichkeit für die Studierenden geschaffen ihre Kenntnisse in der Programmierung von Datenbanktriggern mit PL/SQL zu prüfen.

Glossar

eLML

eLML ist die Abkürzung für eLesson Markup Language, entwickelt und verwendet an der Universität Zürich mit dem Ziel, eLearning-Inhalte zu erstellen und plattformunabhängig zu halten.

CSS

CSS steht für Cascading Style Sheets und ist eine deklarative Stylesheet Sprache, die vor allem im Zusammenhang mit HTML verwendet wird.

Deployment Descriptor

Als Deployment Descriptor bezeichnet man, in Java Web-Anwendungen, die Datei web.xml, welche sich im Unterverzeichnis WEB-INF befindet. Sie stellt eine Konfigurationsdatei für den spezifischen Bereitstellungsprozess dar.

DTD

DTD steht für Document Type Definition und definiert einen Satz von Regeln zur Deklarierung von bestimmten Dokumenttypen.

JavaScript

Javascript ist eine Skriptsprache die innerhalb eines Webbrowsers clientseitig ausgeführt wird.

JDBC

Die Abkürzung JDBC steht für Java Database Connectivity und ist eine Datenbankschnittstelle für die Java Plattform, sie stellt eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller dar.

JNDI

JNDI bedeutet Java Naming and Directory Interface und ist eine Schnittstelle die innerhalb von Java Anwendungen Namensdienste zur Verfügung stellt. Durch sie können Daten und Objektreferenzen anhand eines Namens angelegt und von Nutzern der Schnittstelle abgerufen werden.

SQL

SQL ist die Abkürzung von Structured Query Language und stellt eine Datenbanksprache dar, die von fast allen gängigen relationalen Datenbanken unterstützt wird.

WYSIWYG

WYSIWYG ist das Akronym des „What You See Is What You Get“ Prinzips, hierbei wird ein Dokument während der Bearbeitung auf dem Bildschirm genau so angezeigt, wie es nachher bei der Ausgabe auf einem anderen Gerät aussieht, beispielsweise nach dem Ausdruck durch einen Drucker.

XHTML

XHTML steht für Extensible HyperText Markup Language, sie ist eine textbasierte Auszeichnungssprache zur Strukturierung und semantischen Auszeichnung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten. Sie stellt eine Erweiterung von HTML dar.

XSD

XSD steht für XML Schema Definition, sie wird zur Definition von XML Strukturen verwendet.

XSLT

XSLT bedeutet XSL Transformation, sie ist eine Programmiersprache zur Transformation von XML-Dokumenten und ist Teil der Extensible Stylesheet Language (XSL).

Quellenverzeichnis

[Faeskorn-Woyke , 2009]

Datenbanken und Informationssysteme (DuI) im Verbundstudium Einheit 7: Die Datenbanksprache PL/SQL und aktive Datenbanken

[Ullenboom, 2009]

JavaInsel Ullenboom, Christian; Java ist auch eine Insel; 8. Auflage 2009

[Bleisch, Fisler, 2010]

Bleisch, Susanne, Fisler, Joël, 2010. eLML - eLesson Markup Language, Version vom 04.05.2010, Aus: http://www.elml.org/website/en/text/website.pdf am 05.03.2010

[Gerson, 2000]

Gerson, S.M, 2000, E-CLASS: Creating a Guide to Online Course Development For Distance Learning Faculty. Online Journal of Distance Learning Administration, Volumen III, Number IV, Aus: http://www.westga.edu/~distance/ojdla/winter34/gerson34.html am 20.04.2011

[Java BluePrints, 2002]

O.V. , Java BluePrints Model-View-Controller, Aus: http://java.sun.com/blueprints/patterns/MVC-detailed.html am 13.04.2011

[Mahmoud, 2003]

Qusay H. Mahmoud, 2003, Servlets and JSP Pages Best Practices, http://www.oracle.com/technetwork/articles/javase/servlets-jsp-140445.html

[Gamma, 1995]

Gamma, Helm, Johnson, and Vlissides, 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*.

[Hunter, Crawford, 1998]

Jason Hunter, William Crawford, 1998, Java Servlet Programming

[Steyer, 2006]

Ralph Steyer, 2006, AJAX mit Java-Servlets und JSP

[Wenz, O.J.]

Christian Wenz, JavaScript und AJAX

Anhang

5.1 Aufgabenkatalog

Aufgabe 1:

Aufgabentext:
Die Summer aller Gehälter darf ein bestimmtes Budget nicht überschreiten.
Es soll ein Trigger "trg_budget_check" erstellt werden, der bei Hinzufügen eines neuen Mitarbeiters in der Tabelle "Angestellte", bei Überschreitung eines Gesamtbudgets von 100.000€, alle Gehälter höher als 5000€, auf 5000€ kürzt

Auslösende Aktion:
INSERT INTO Angestellte VALUES (28, 3, 'Fehlt noch', 'Schreibkraft','Budar', 'Hermann', 'm', TO_Date('01/01/1996 12:00:00 AM', 'MM/DD/YYYY HH:MI:SS AM'), 2800, 1500, 'Dortmund', 'Planckstr.', NULL)

Musterlösung:
```
create or replace TRIGGER trg_budget_check
AFTER INSERT ON Angestellte
DECLARE
  v_sum NUMBER;
BEGIN
SELECT SUM(gehalt) INTO v_sum FROM Angestellte;
IF v_sum > 100000 THEN
UPDATE Angestellte SET gehalt=5000 where gehalt > 5000;
END IF;
END;
```

Aufgabe 2:

Aufgabentext:
Abteilungsleiter müssen einer Mindestgehaltsgruppe angehören.
Bei Hinzufügen einer neuen Abteilung oder wenn sich der Leiter einer Abteilung ändert, soll ,durch den Trigger "trg_leiter_gehalt", geprüft werden ob das Gehalt des Abteilungleiters mindestens in die Gehaltsgruppe 3 fällt. Ist das Gehalt zu niedrig, so soll es auf das Mindestgehalt dieser Gehaltsgruppe erhöht werden.

Auslösende Aktionen:
1. update Abteilungen set Leiter=10 where Name='Einkauf';
2. update Abteilungen set Leiter=12 where Name='Produktion';

Musterlösung:

```
create or replace TRIGGER trg_leiter_gehalt
AFTER INSERT OR UPDATE ON ABTEILUNGEN
REFERENCING OLD AS alt NEW AS neu
FOR EACH ROW
DECLARE
  geh number;
  gehmin number;
BEGIN
  select gehalt into geh from angestellte where ang_nr=:neu.leiter;
  select min_gehalt into gehmin from geh_klassen where geh_klasse=3;
  if (geh < gehmin) then
    update angestellte set gehalt=gehmin where ang_nr=:neu.leiter;
  END IF;
END;
```

Aufgabe 3:

Aufgabentext:
Folgeverarbeitung bei Bestandsänderung.
Wenn sich der Bestand eines Teils in der Tabelle Lagerbestand ändert, soll der Bestand in der Tabelle Teile automatisch mit angepasst werden. Der Trigger soll unter dem Namen "trg_bestand_update" erstellt werden

Auslösende Aktion:

```
update Lagerbestand set bestand = 1000 where tnr=1 and lanr=2;
```

Musterlösung:

```
CREATE OR REPLACE TRIGGER trg_bestand_update
AFTER UPDATE OF Bestand ON Lagerbestand
FOR EACH ROW
BEGIN
        UPDATE Teile
        SET Bestand =:NEW.Bestand -:OLD.Bestand + Bestand
        WHERE TNr =:NEW.TNr;
END;
```

Aufgabe 4:

Aufgabentext:
Folgeverarbeitung vollständig bezahlter Rechnungen.
Der Trigger "trg_umsatz_update" soll, sobald ein Kunde den Rechnungsbetrag eines Auftrags vollständig gezahlt hat, in der Tabelle Artikel die im Auftrag umgesetzten Mengen zum Jahresumsatz hinzu Addieren

Auslösende Aktionen:

1. update Auftraege set bereits_gezahlt=10500 where auftragsnr=1;
2. update Auftraege set bereits_gezahlt=10000 where auftragsnr=2;

Musterlösung:

```
CREATE OR REPLACE TRIGGER trg_umsatz_update
BEFORE INSERT OR UPDATE ON AUFTRAEGE
REFERENCING OLD AS alt NEW AS neu
FOR EACH ROW
DECLARE
rechnungssumme number;
BEGIN
  if (:alt.bereits_gezahlt <> :neu.bereits_gezahlt) then
    select SUM(Gesamt) into rechnungssumme from (select
auftragspositionen.tnr,auftragspositionen.menge,artikel.verkaufspreis,
    (auftragspositionen.menge*artikel.verkaufspreis) as gesamt  from
auftragspositionen inner join
    artikel on auftragspositionen.tnr=artikel.tnr where
auftragspositionen.auftragsnr=:neu.auftragsnr);
    if (rechnungssumme = :neu.bereits_gezahlt) then
      update artikel set jahresumsatz=jahresumsatz + (select menge from
auftragspositionen where AUFTRAGSNR=:neu.auftragsnr
      and auftragspositionen.TNR = Artikel.TNR) where tnr in (select tnr from
auftragspositionen where auftragsnr=:neu.auftragsnr);
    end if;
  end if;
END;
```

Aufgabe 5:

Aufgabentext:

Gehaltserhöhungen müssen mindestens in einer Höhe von 10% erfolgen.
Bei Erhöhung des Gehalts eines Angestellten soll durch den Trigger "trg_erh_gehalt" geprüft werden, ob diese Regelung eingehalten wurde. Sollte der neue Wert zu niedrig sein, soll das Gehalt des Angestellten automatisch um dieses Minimum erhöht werden.

Auslösende Aktionen:

1. update angestellte set gehalt=4100 where ang_nr=26;
2. update angestellte set gehalt=6000 where ang_nr=6;

Musterlösung:

```
CREATE OR REPLACE TRIGGER trg_erh_gehalt
BEFORE UPDATE ON ANGESTELLTE
REFERENCING OLD AS alt NEW AS neu
FOR EACH ROW
DECLARE
erh number;
erh_min number;
BEGIN
  if (:alt.gehalt < :neu.gehalt) then
    erh := :neu.gehalt - :alt.gehalt;
    erh_min := :alt.gehalt * 0.1;
    if (erh < erh_min) then
      :neu.gehalt := :alt.gehalt + erh_min;
```

```
    end if;
  end if;
END;
```

Aufgabe 6:

Aufgabentext:
Archivierung gelöschter Auftragspositionen.
Wenn ein Datensatz aus der Tabelle Auftragspositionen gelöscht wird, sollen die gelöschten Datensätze, durch den Trigger "trg_pos_archiv" ,in der Tabelle Positionsarchiv archiviert werden.

Auslösende Aktion:
```
delete from Auftragspositionen where AuftragsNR=5;
```

Musterlösung:
```
CREATE OR REPLACE TRIGGER trg_pos_archiv
AFTER DELETE ON AUFTRAGSPOSITIONEN
REFERENCING OLD AS alt
FOR EACH ROW
BEGIN
  insert into POSITIONSARCHIV values (:alt.auftragsnr, :alt.tnr, :alt.menge);
END;
```

Aufgabe 7:

Aufgabentext:
Protokollierung von Gehaltsänderungen.
Wenn in der Tabelle Angestellte Änderungen in der Spalte "Gehalt" vorgenommen werden, soll die betreffende Angestellten Nr, die Gehaltsveränderung d.h. altes Gehalt und neues Gehalt sowie das aktuelle Tagesdatum, durch den Trigger "trg_geh_prot", in der Tabelle "Gehaltsprotokoll" protokolliert werden.
Info: Aktuelles Tagesdatum über TO_CHAR(sysdate) einfügen.

Auslösende Aktionen:
1. update Angestellte set Gehalt=5500 where Ang_Nr=7;
2. update Angestellte set Nachname='Schmitz' where Ang_Nr=8;

Musterlösung:
```
CREATE OR REPLACE TRIGGER trg_geh_prot
AFTER UPDATE ON ANGESTELLTE
REFERENCING OLD AS alt NEW AS neu
FOR EACH ROW
BEGIN
  if (:alt.gehalt <> :neu.gehalt) then
    insert into GEHALTSPROTOKOLL values (:alt.ang_nr, :alt.gehalt , :neu.gehalt ,
TO_CHAR(sysdate));
  end if;
END;
```

Aufgabe 8:

Aufgabentext:
Gehälter dürfen nicht sinken.
Durch den Trigger "trg_geh_check" soll sichergestellt werden, dass in der Spalte Gehalt in der Tabelle "Angestellte" kein niedrigerer Wert als der bisherige eingetragen wird. In diesem Fall soll eine Ausnahme mit der ID:-20001 und dem Text:"Gehälter dürfen nicht sinken" ausgelöst werden.

Auslösende Aktionen:
1. update Angestellte set gehalt=8000 where ang_nr=3;
2. update Angestellte set gehalt=9000 where ang_nr=1;

Musterlösung:
```
CREATE OR REPLACE TRIGGER trg_geh_check
BEFORE UPDATE ON ANGESTELLTE
REFERENCING OLD AS alt NEW AS neu
FOR EACH ROW
BEGIN
  if (:neu.gehalt < :alt.gehalt) then
    RAISE_APPLICATION_ERROR(-20001,'Gehälter dürfen nicht sinken');
  end if;
END;
```

Aufgabe 9:

Aufgabentext:
Personalnummern dürfen nicht geändert werden.
Durch den Trigger "trg_ang_nr_check" soll verhindert werden, dass die Spalte "Ang_Nr" in der Tabelle Angestellte nachträglich geändert wird. In diesem Fall soll eine Ausnahme mit der ID:-20001 und dem Text:"Personalnummern dürfen nicht geändert werden" ausgelöst werden.

Auslösende Aktionen:
1. update Angestellte set Nachname='Schmitz' where Ang_Nr=8;
2. update angestellte set ang_nr=30 where ang_nr=2;

Musterlösung:
```
CREATE OR REPLACE TRIGGER trg_ang_nr_check
BEFORE UPDATE ON ANGESTELLTE
REFERENCING OLD AS alt NEW AS neu
FOR EACH ROW
BEGIN
  if (:neu.ang_nr <> :alt.ang_nr) then
    RAISE_APPLICATION_ERROR(-20001,'Personalnummern dürfen nicht geändert
werden');
  end if;
END;
```

Aufgabe 10:

Aufgabentext:
Gehaltsklassen dürfen sich nicht überschneiden.
Der Trigger "trg_geh_klassen" soll beim Einfügen/Ändern eines Datensatzes in der Tabelle geh_klassen die neuen bzw. geänderten Zeilen prüfen, ob sich hierdurch Überschneidungen zu anderen Gehaltsklassen ergeben. Sollte dies der Fall sein soll der Trigger eine Exception mit der ID: -20001 und dem Text: "Gehaltsklassen dürfen sich nicht überschneiden" auslösen.
(Tip: Verwenden Sie zur Lösung einen autonomen Row-Trigger)

Auslösende Aktionen:

1. update Geh_klassen set max_gehalt=1500 where geh_klasse=5;
2. update Geh_klassen set max_gehalt=4500 where geh_klasse=4;

Musterlösung:

```
CREATE OR REPLACE TRIGGER trg_geh_klassen
  BEFORE INSERT OR UPDATE
  ON geh_klassen
  FOR EACH ROW
DECLARE
  v_anzahl   NUMBER (5) DEFAULT 0;
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  SELECT   COUNT ( * )
    INTO   v_anzahl
    FROM   geh_klassen
   WHERE   (:NEW.min_gehalt BETWEEN min_gehalt AND max_gehalt
           OR :NEW.max_gehalt BETWEEN min_gehalt AND max_gehalt);
  COMMIT;
  IF v_anzahl > 1
  THEN
     RAISE_APPLICATION_ERROR (
       -20001,'Gehaltsklassen dürfen sich nicht überschneiden');
  END IF;
END;
```

Weitere Anhänge befinden sich in elektronischer Form auf der beiliegenden CD.

- Diplomarbeit als ODF-Datei
- Diplomarbeit als PDF-Datei
- Eclipse Projekte

Erklärung über die selbstständige Abfassung der Arbeit

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zur haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Frechen, den 30.05.2011 _____

(Markus Rechs)